

## **Project Info:**

Mac (Carbon PEF) App Name: Eschaton  
Mac (Carbon Mach-O) App Name: Eschaton  
Mac (Classic) App Name: Eschaton  
Windows App Name: Eschaton.exe  
Linux App Name: Eschaton  
Long Version: 0.8  
Major Version: 1  
Minor Version: 0  
Sub Version: 0  
Release: 0  
Non-Release: 0  
Mac Creator Code: esch  
Windows MDI Caption:  
Minimum Memory Size: 2048  
Standard Memory Size: 4096

## **Class App**

Inherits Application

Const kCloseWindow = "&Close

"

Const kEditClear = "&Delete"

Const kFileQuit = "&Quit"

Const kFileQuitShortcut = ""

## **App.OpenDocument:**

Sub OpenDocument(item As FolderItem)

select case item.Type

case H1\_Filetypes.HaloMapFile.Name

if WindowCount > 0 then

if Window(0) isa Main\_Window then

//top most window is a main window so just add the map to it

Main\_Window(Window(0)).OpenLite(item)

else

//not a main window so don't use it

end

else

//no windows open

dim w as new Main\_Window

w.OpenLite(item)

end

end select

End Sub

## **App.AboutMenu:**

Function AboutMenu() As Boolean

```
dim w as new About_Window
w.show
Return True
```

End Function

### **App.FileLoadTag:**

```
Function FileLoadTag() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.EschatonMetaFile)
    if f = nil then return true
```

```
    dim w as new Main_Window
    w.load_tag(f)
    w.show
```

```
    Return True
```

End Function

### **App.FileLoadTagsRecursively:**

```
Function FileLoadTagsRecursively() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.EschatonMetaFile)
    if f = nil then return true
    dim folder_f as FolderItem = selectFolder_custom("Top Level Folder",_
    "Select the top level folder for the extracted meta data")
    if folder_f = nil then return true
    if f.urlpath.instr(folder_f.urlpath) = 0 then
        errorbox("Error: Selected meta is not within selected folder")
        return true
    end
```

```
    dim w as new Main_Window
    w.load_tags_recursively(f, folder_f)
    w.show
```

```
    return true
```

End Function

### **App.FileQuit:**

```
Function FileQuit() As Boolean
    Quit
```

```
    Return True
```

End Function

### **App.NewWindow:**

```
Function NewWindow() As Boolean
    dim w as new Main_Window
    w.Show
    Return True
End Function
```

### **App.Open:**

```

Function Open() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
    if f = nil then Return true
    dim w as new Main_Window
    w.Open(f)
    w.Show
    Return True
End Function

```

### **App.OpenLite:**

```

Function OpenLite() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
    if f = nil then Return true
    dim w as new Main_Window
    w.OpenLite(f)
    w.Show
    Return True
End Function

```

### **App.OpenPreferences:**

```

Function OpenPreferences() As Boolean
    dim w as new Preferences_Window
    w.show

    Return True
End Function
End Class

```

## **Module FileManip**

### **FileManip.errorbox:**

```

Sub errorbox(message as string)
    Dim d as New MessageDialog //declare the MessageDialog object
    Dim b as MessageDialogButton //for handling the result
    d.icon=MessageDialog.GraphicCaution //display warning icon
    d.ActionButton.Caption="Ok"
    d.Message="Error"
    d.Explanation=message

    b=d.ShowModal //display the dialog
End Sub

```

### **FileManip.generate\_folders:**

```

Function generate_folders(f as folderitem, input_string as string) As folderItem
    if f.Exists = false then return nil
    if f.Directory = false then return nil

    dim folders(-1) as string
    folders = input_string.split("\")
    for i as integer = 0 to folders.ubound

```

```

    if folders(i) = "" then folders(i) = "BLANK"
next

dim out_f as FolderItem = f

for i as integer = 0 to ubound(folders) - 1
    out_f = out_f.child(folders(i))
    //if out_f = nil then
    out_f.CreateAsFolder
    //end
next

return out_f
End Function

```

### **FileManip.getOpenFolderitem\_custom:**

Function getOpenFolderitem\_custom(title as string, prompt as string, initial\_directory as folderitem = nil, filter as string = "") As folderitem

```

    Dim open_dialog as New OpenFileDialog
    open_dialog.ActionButtonCaption="Select"
    open_dialog.Title=title
    open_dialog.PromptText=prompt
    if initial_directory <> nil then
        open_dialog.InitialDirectory= initial_directory
    end
    if filter <> "" then
        open_dialog.Filter=filter
    end
    return open_dialog.ShowDialog
End Function

```

### **FileManip.remove\_redundant\_ints:**

Function remove\_redundant\_ints(input() as integer) As integer()

```

    for i as integer = 0 to UBound(input)
        while (i+1 <= UBound(input)) AND (input.IndexOf(input(i), i+1) > i)
            dim ind as integer = input.IndexOf(input(i), i+1)
            input.Remove ind
        wend
    next

    return input

```

```

//bad implementation
'dim temp(-1) as integer
'for i as integer = 0 to UBound(input)
'temp.Append i
'next
'
'input.SortWith(temp)
'
'//removes all later versions of the index
'for i as integer = 0 to UBound(input)
'while (i+1 <= UBound(input)) AND (input.IndexOf(input(i), i+1) > i)

```

```

'dim ind as integer = input.IndexOf(input(i), i+1)
'input.Remove ind
'temp.Remove ind
'wend
'next
'
'temp.SortWith(input)
'
'return input

```

End Function

### **FileManip.remove\_redundant\_ints\_other:**

Function remove\_redundant\_ints\_other(base\_ar() as integer, extra\_ar() as integer) As integer()

//removes from extra\_ar everything that is in base\_ar

```

for i as integer = 0 to UBound(base_ar)
    dim temp as integer = base_ar(i)
    while extra_ar.IndexOf(temp) >= 0
        extra_ar.remove(extra_ar.IndexOf(temp))
    wend
next

```

return extra\_ar

End Function

### **FileManip.reverse:**

Function reverse(input as string) As string

//this function reverses the characters it receives (which it will call input) and reverses the order

//find the length of the string passed

dim i as integer = Len(input)

dim output as string

//start from the last character in the string and write it to the string output

//then decrement i, to read the next earlier character

while i >=0

//the mid function starts from the i-th character, and gets in this case 1 character

output = output + input.mid(i,1)

i=( i - 1)

wend

//this returns the reversed string to anything calling it

//this statement is extremely important for all you non-programmers out there

return output

End Function

### **FileManip.reverse:**

Function reverse(extends input as string) As string

//this function reverses the characters it receives (which it will call input) and reverses the order

//find the length of the string passed

dim i as integer = Len(input)

dim output as string

```

//start from the last character in the string and write it to the string output
//then decrement i, to read the next earlier character
while i >=0
    //the mid function starts from the i-th character, and gets in this case 1 character
    output = output + input.mid(i,1)
    i=( i - 1)
wend

//this returns the reversed string to anything calling it
//this statement is extremely important for all you non-programmers out there
return output
End Function

```

### FileManip.scaleit:

Function scaleit(inpic As picture,scale As double) As picture

\*\*\*\*\*

'Scaleit used to scale down images Quickly and works on windows

'Just pass a picture and the scale and the scaled picture is passed back

'By Luke Jackson

'modified-corrected from corrupted webpost by Altimit01

\*\*\*\*\*

dim h,w,hc,wc,sh,sw As integer

dim col as color

dim scaled As picture

dim prgb,srgb As rgbSurface

dim scaleddepth as integer

h= inpic.height

w = inpic.width

sh = max((h\*scale), 1)

sw = max((w\*scale), 1)

scaled = newpicture(sw,sh,32)

prgb = inpic.rgbSurface

srgb = scaled.rgbSurface

for hc = 0 to sh

for wc = 0 to sw

srgb.pixel(wc,hc) = prgb.pixel(floor(wc\*(1/scale)),floor(hc\*(1/scale)))

next

next

return scaled

exception

errorBox "Error in Scaleit Win32"

End Function

### FileManip.scaleit:

Function scaleit(extends inpic As picture,scale As double) As picture

\*\*\*\*\*

'Scaleit used to scale down images Quickly and works on windows

'Just pass a picture and the scale and the scaled picture is passed back

'By Luke Jackson

'modified-corrected from corrupted webpost by Altimit01

\*\*\*\*\*

```

dim h,w,hc,wc,sh,sw As integer
dim col as color
dim scaled As picture
dim prgb,srgb As rgbSurface
dim scaleddepth as integer
h= inpic.height
w = inpic.width
sh = max((h*scale), 1)
sw = max((w*scale), 1)
scaled = newpicture(sw,sh,32)
prgb = inpic.rgbSurface
srgb = scaled.rgbSurface
for hc = 0 to sh
    for wc = 0 to sw
        srgb.pixel(wc,hc) = prgb.pixel(floor(wc*(1/scale)),floor(hc*(1/scale)))
    next
next
return scaled

```

```

exception
    errorBox "Error in Scaleit Win32"
End Function

```

### **FileManip.scaleit\_max:**

Function scaleit\_max(inpic As picture, max\_dimension as integer) As picture

```

dim h,w As integer
dim scale as double
h= inpic.height
w = inpic.width
if w > h then
    dim w_d as double = w
    dim max_d as double = max_dimension
    scale = max_d/w_d
else
    dim h_d as double = h
    dim max_d as double = max_dimension
    scale = max_d/h_d
end

```

```

    Return scaleit(inpic, scale)
End Function

```

### **FileManip.scaleit\_max:**

Function scaleit\_max(extends inpic As picture, max\_dimension as integer) As picture

```

dim h,w As integer
dim scale as double
h= inpic.height
w = inpic.width
if w > h then
    dim w_d as double = w
    dim max_d as double = max_dimension
    scale = max_d/w_d
else

```

```

    dim h_d as double = h
    dim max_d as double = max_dimension
    scale = max_d/h_d
end

```

```

    Return scaleit(inpic, scale)
End Function

```

### **FileManip.selectFolder\_custom:**

Function selectFolder\_custom(title as string, prompt as string, initial\_directory as folderItem = nil) As folderItem

```

    Dim dlg as New SelectFolderDialog
    dlg.ActionButtonCaption = "Select"
    dlg.Title = title
    dlg.PromptText = prompt
    if initial_directory <> nil then
        dlg.InitialDirectory= initial_directory
    end
    return dlg.ShowModal()
End Function

```

### **FileManip.xml\_to\_string:**

Function xml\_to\_string(input as string) As string

```

    //this takes an xml file as an input string and makes it human readable
    dim temp_string as string = input.ReplaceAll("><", ">" + EndOfLine + "<")
    dim temp_split() as string = temp_string.Split(EndOfLine)
    dim indent_count as integer = 0
    //chr(9) = horizontal tab
    for i as integer = 0 to temp_split.ubound
        select case temp_split(i).Mid(1,2)
            case "<?"
                //xml header, ignore
            case "<!"
                //xml comment, ignore
            case "</"
                //xml tag close, decrement indent_count
                indent_count = max(indent_count - 1, 0)
            case else
                //xml element, indent
                for j as integer = 1 to indent_count
                    temp_split(i) = chr(9) + temp_split(i)
                next
            end select
        select case temp_split(i).Right(2)
            case "?>"
                //xml header, ignore
            case "->"
                //xml comment, ignore
            case "/>"
                //xml element, ignore
            case else
                if temp_split(i).Mid(1,2) <> "</" then
                    //so it's not a tag closure
                    indent_count = indent_count + 1
                end
            end select
        end select
    next i
    return temp_string
End Function

```



```

        end select
    next
    dim output as string = ""
    for i as integer = 0 to temp_split.Ubound
        output = output + temp_split(i) + EndOfLine
    next
    return output
End Function
End Module

```

## **Class SmartSplitter**

Inherits Canvas

```

Private Const enableDebugCanvasCode = True
Const SmartSplitter_Version = 1.32
Event Close()
Sub ()
Event MouseDown(X as Integer, Y as Integer)
Sub ()
Event MouseDrag(X as Integer, Y as Integer)
Sub ()
Event MouseEnter()
Sub ()
Event MouseUp(X as Integer, Y as Integer)
Sub ()
Event Moved(Distance as Integer)
Sub ()
Event Open()
Sub ()
Event Paint(g as Graphics)
Sub ()

```

### **SmartSplitter.Close:**

```

Sub Close()
    ' dispose of controls before closing splitter for a cleaner close

    unAttachAllControls

    #if enableDebugCanvasCode then
        debugCanvas = nil
    #endif

    Close ' call sub's Close Event
End Sub

```

### **SmartSplitter.MouseDown:**

```

Function MouseDown(X As Integer, Y As Integer) As Boolean

    if conceal then return false

```

assignLimits

mouseDownTriggered = true  
mouseDownX = X  
mouseDownY = Y

positionAtMouseDown = top  
if isVertical then positionAtMouseDown = left

refreshDebugCanvas

MouseDown(X, Y) ' call sub's MouseDown Event (minus the boolean)  
return true

End Function

### **SmartSplitter.MouseDrag:**

Sub MouseDrag(X As Integer, Y As Integer)

'assignLimits

if isVertical then  
    moveDistance X – mouseDownX  
else  
    moveDistance Y – mouseDownY  
end if

#if enableDebugCanvasCode then  
    if debugCanvas <> nil then debugCanvas.markSplitter me, minLimit, maxLimit, isVertical  
#endif

MouseDrag X, Y ' call sub's MouseDrag Event  
End Sub

### **SmartSplitter.MouseEnter:**

Sub MouseEnter()  
    assignCursor

MouseEnter ' call sub's MouseEnter Event  
End Sub

### **SmartSplitter.MouseUp:**

Sub MouseUp(X As Integer, Y As Integer)  
    dim i, d as integer

if DisableLiveDrag then  
    d = position – positionAtMouseDown  
    adjustAttachedControls d  
    Moved d  
end if

mouseDownTriggered = false

```
if DisableLiveDrag then window.refresh true
```

```
refreshDebugCanvas
```

```
MouseUp X, Y ' call sub's MouseUp Event
```

```
End Sub
```

### **SmartSplitter.Open:**

```
Sub Open()
```

```
initializeSmartSplitter ' this method calls sub's Open Event
```

```
End Sub
```

### **SmartSplitter.Paint:**

```
Sub Paint(g As Graphics)
```

```
drawHandle
```

```
Paint g ' call sub's Paint Event
```

```
End Sub
```

### **SmartSplitter.adjustAttachedControls:**

```
Private Sub adjustAttachedControls(d as integer, fromBehave as boolean = false)
```

```
dim i, pos, size as integer
```

```
dim s as string
```

```
dim vis as boolean
```

```
for i = 0 to ubound(attachedControl)
```

```
if attachedControl(i) <> nil then
```

```
if isVertical then
```

```
pos = attachedControl(i).left
```

```
size = attachedControl(i).width
```

```
if attachedControlBehind(i) then ' it is on the left
```

```
if attachedControlShouldCenter(i) then
```

```
pos = minLimit + width/2 + (left - minLimit)/2 - attachedControl(i).width/2
```

```
else
```

```
if attachedControlShouldMove(i) then pos = pos + d
```

```
if attachedControlShouldResize(i) then size = size + d
```

```
end if
```

```
else ' it is on the right
```

```
if attachedControlShouldCenter(i) then
```

```
pos = left + width + (maxLimit - left - width)/2 - attachedControl(i).width/2
```

```
else
```

```
if attachedControlShouldMove(i) then pos = pos + d
```

```
if attachedControlShouldResize(i) then size = size - d
```

```
end if
```

```
end if
```

```
#pragma disableBackgroundTasks
```

```
if size < 1 and attachedControl(i).visible then attachedControl(i).visible = false
```

```
if attachedControl(i).width <> size then attachedControl(i).width = size
```

```
if attachedControl(i).left <> pos then attachedControl(i).left = pos
```

```

else ' is horizontal
    pos = attachedControl(i).top
    size = attachedControl(i).height
    if attachedControlBehind(i) then ' it is above
        if attachedControlShouldCenter(i) then
            pos = minLimit + height/2 + (top - minLimit)/2 - attachedControl(i).height/2
        else
            if attachedControlShouldMove(i) then pos = pos + d
            if attachedControlShouldResize(i) then size = size + d
        end if
    else ' it is below
        if attachedControlShouldCenter(i) then
            pos = top + height + (maxLimit - top - height)/2 - attachedControl(i).height/2
        else
            if attachedControlShouldMove(i) then pos = pos + d
            if attachedControlShouldResize(i) then size = size - d
        end if
    end if
    #pragma disableBackgroundTasks
    if size < 1 and attachedControl(i).visible then attachedControl(i).visible = false
    if attachedControl(i).height <> size then attachedControl(i).height = size
    if attachedControl(i).top <> pos then attachedControl(i).top = pos
end if

vis = (size >= attachedControlMinSizeVisible(i))

if size < 1 then vis = false ' hide control if it has a negative or 0 width or height to avoid display glitches

' hide control if this splitter is overlapping or on the wrong side
if (not attachedControlShouldMove(i) and not attachedControlShouldResize(i)) then
    if isVertical then
        if attachedControlBehind(i) then ' it should be on the left
            if left+1 < attachedControl(i).left+attachedControl(i).width then vis = false
        else ' it should be on the right
            if left+width-1 > attachedControl(i).left then vis = false
        end if
    else ' is horizontal
        if attachedControlBehind(i) then ' it should be above
            if top+2 < attachedControl(i).top+attachedControl(i).height then vis = false
        else ' it should be below
            if top+height-2 > attachedControl(i).top then vis = false
        end if
    end if
end if

' hide controls that are outside of their parent control
if not attachedControl(i) isA smartSplitter and attachedControl(i).parent <> nil then ' (nil if parent is
window)
    if attachedControl(i).left < rectControl(attachedControl(i).parent).left or attachedControl(i).left
+attachedControl(i).width > rectControl(attachedControl(i).parent).left
+rectControl(attachedControl(i).parent).width then vis = false

```

```

    if attachedControl(i).top < rectControl(attachedControl(i).parent).top or attachedControl(i).top
    +attachedControl(i).height > rectControl(attachedControl(i).parent).top
    +rectControl(attachedControl(i).parent).height then vis = false
end if

if vis then
    if not attachedControl(i).visible and attachedControlNormallyVisible(i) then attachedControl(i).visible =
    true
else
    if attachedControl(i).visible then attachedControl(i).visible = false
end if

#if enableDebugCanvasCode then
    if debugCanvas <> nil then debugCanvas.markControl attachedControl(i)
#endif

end if 'attachedControl(i) <> nil

next

if targetWin32 and not disableLiveDrag and not debug then refresh

' now tell any attached smartSplitters with same orientation to behave so they will stay snapped
for i = 0 to ubound(attachedControl)
    if attachedControl(i) isA smartSplitter and not fromBehave then
        if isVertical = smartSplitter(attachedControl(i)).isVertical then smartSplitter(attachedControl(i)).behave
    end if
end for
next

```

End Sub

### SmartSplitter.assignCursor:

Private Sub assignCursor()

```

if conceal or not enabled then
    mouseCursor = nil
else
    if isVertical then
        #if RBVersion > 6 then
            mouseCursor = System.Cursors.SplitterEastWest
        #else
            #if targetWin32 then
                mouseCursor = cursorHorizontalAdjustWin32
            #else
                mouseCursor = cursorHorizontalAdjust
            #endif
        #endif
    #endif
    'For ERROR – cursor property does not exist:
    ' To use with REALbasic 5.x, cursor files will need to be imported into this project.
    ' One way to accomplish this in REALbasic 5.x is to drag the "SmartSplitter" folder
    ' out of the example project "SmartSplitter_Tester.rb" project window. Then drag
    ' the folder onto the project window of this project.
    '
    ' Another way is to import the cursors contained in the compressed file,

```

```

' "Cursors for REALbasic 5.x.sit", included in the original distribution folder.
' Once the files are uncompressed, they can be dragged into this project.
' NOTE: These cursors are contained in the Macintosh resource fork of each file -
'     so this might only be possible using Mac OS.
else
  #if RBVersion > 6 then
    mouseCursor = System.Cursors.SplitterNorthSouth
  #else
    #if targetWin32 then
      mouseCursor = cursorVerticalAdjustWin32
    #else
      mouseCursor = cursorVerticalAdjust
    #endif
  #endif
end if
end if

```

End Sub

### **SmartSplitter.assignLimits:**

Private Sub assignLimits()

```

if minLimit = 9999 then minLimit = 0

```

```

if isVertical then

```

```

  if minAttachedControl <> nil then
    proportionMin = minAttachedControl.left - width
    if not minLimitPropSet then minLimit = proportionMin
  end if

```

```

  if maxAttachedControl <> nil then
    proportionMax = maxAttachedControl.left + maxAttachedControl.width
    if not maxLimitPropSet then maxLimit = proportionMax
  else
    proportionMax = maxLimit + window.width - windowSizeLast
    if not maxLimitPropSet then maxLimit = proportionMax
  end if
  windowSizeLast = window.width

```

```

else

```

```

  if minAttachedControl <> nil then
    proportionMin = minAttachedControl.top - height
    if not minLimitPropSet then minLimit = proportionMin
  end if

```

```

  if maxAttachedControl <> nil then
    proportionMax = maxAttachedControl.top + maxAttachedControl.height
    if not maxLimitPropSet then maxLimit = proportionMax
  else
    proportionMax = maxLimit + window.height - windowSizeLast
    if not maxLimitPropSet then maxLimit = proportionMax
  end if

```

```
end if
windowSizeLast = window.height
```

```
end if
```

```
End Sub
```

### **SmartSplitter.attach:**

```
Sub attach(c as rectControl)
```

```
    ' Attaches a control to the splitter. If this is done in the SmartSplitter's
```

```
    ' Open method, then attachNearbyControls is not automatically called.
```

```
    dim i, edge as integer
```

```
    dim behind, move, resize, embedded as boolean
```

```
    initializeSmartSplitter
```

```
    edge = 5 ' how close a control needs to be for it to be set to resize when appropriate
```

```
    if c = nil then return
```

```
    if c = self then return ' never attach self - things would get very ugly
```

```
    if c isA line then return
```

```
    ' check if c is already attached or is embedded in a control that is already attached
```

```
    for i = 0 to ubound(attachedControl)
```

```
        if c.parent = attachedControl(i) then embedded = true ' control being attached is embedded in another
        attached Control
```

```
        if c = attachedControl(i) then
```

```
            if showWarningsDialogs then msgBox "The control "" + c.name + "" is already attached to smartSplitter "" +
            name + ""." This attach request will be ignored."
```

```
            return
```

```
        end if
```

```
    next
```

```
    'if embedded = true then
```

```
    'for i = 0 to ubound(attachedControl)
```

```
    'if rectControl(c.parent).parent = attachedControl(i) then embedded = false ' control being attached is embedded
    in another attached Control
```

```
    'next
```

```
    'end if
```

```
    attachedControl.append c
```

```
    attachedControlNormallyVisible.append c.visible
```

```
    attachedControlShouldCenter.append false
```

```
    attachedControlMinSizeVisible.append suggestMinSizeControlShouldBeVisible(c)
```

```
    if isVertical then
```

```
        attachedControlSize.append c.width
```

```
        if c.left < left then ' control is to the left
```

```
            behind = true
```

```
            if embedded then
```

```
                ' if right side same as parent then resize (i.e. popupMenu in a pagePanel)
```

```

        if c.lockLeft and RBVersion<6 or abs(rectControl(c.parent).left + rectControl(c.parent).width - c.left -
        c.width) < edge then
            resize = suggestWhetherControlShouldResize(c)
        end if
    else
        resize = suggestWhetherControlShouldResize(c)
    end if
    SetMinAttachedControl c, true
else ' assume it is to the right
    if embedded then
        if c.lockRight and RBVersion<6 then resize = suggestWhetherControlShouldResize(c)
    else
        resize = suggestWhetherControlShouldResize(c)
        move = true
    end if
    SetMaxAttachedControl c, true
end if
else
    attachedControlSize.append c.height
    if c.top < top then ' control is above
        behind = true
        if embedded then
            if c.lockTop and RBVersion<6 or abs(rectControl(c.parent).top + rectControl(c.parent).height - c.top -
            c.height) < edge then
                resize = suggestWhetherControlShouldResize(c)
            end if
        else
            resize = suggestWhetherControlShouldResize(c)
        end if
        SetMinAttachedControl c, true
    else ' assume it is below
        if embedded then
            if c.lockBottom and RBVersion<6 then resize = suggestWhetherControlShouldResize(c)
        else
            resize = suggestWhetherControlShouldResize(c)
            move = true
        end if
        SetMaxAttachedControl c, true
    end if
end if

attachedControlBehind.append behind
attachedControlShouldMove.append move
attachedControlShouldResize.append resize

' raise offsetSnapTo for larger controls
i = suggestOffsetSizeForControl(c)
if behind then
    if not minLimitOffsetPropSet then minLimitOffset = max(minLimitOffset,i)
else
    if not maxLimitOffsetPropSet then maxLimitOffset = max(maxLimitOffset,i)
end if

```



```

if not doNotAttachEmbeddedControls then ' recursively attach embedded controls
    if controlCommonlyHasEmbeddedControls(c) then
        for i = 0 to window.controlCount-1 ' attach all controls inside this control
            if window.control(i) isA rectControl then
                if rectControl(window.control(i)).parent = c then attach rectControl(window.control(i))
            end if
        next
    end if
end if

```

End Sub

### SmartSplitter.attachNearbyControls:

Sub attachNearbyControls()

```

' Attaches controls with edges within 5 pixels of the edges of the splitter.
' Embedded controls will be recursively added unless you set doNotAttachEmbeddedControls = True.
'dim i, edge as integer
'dim okToAttach as boolean
'dim c as rectControl
'
'initializeSmartSplitter
'
'edge = 5 ' how close a control needs to be for it to auto-attach
'
'// doesn't handle recursively embedded controls
'for i = 0 to window.controlCount - 1
'if window.control(i) isA rectControl then
'c = rectControl(window.control(i))
'if parent = c.parent then
'okToAttach = true
'if parent isA tabPanel and tabPanelIndex <> c.tabPanelIndex then okToAttach = false
'if parent isA pagePanel and tabPanelIndex <> c.tabPanelIndex then okToAttach = false
'
'if okToAttach then
'if isVertical then
'if (c.top > top-edge and c.top < top+height+edge) or (c.top+c.height > top-edge and c.top+c.height < top
+height+edge) or (top > c.top-edge and top+height < c.top+c.height+edge) then
'if abs(left - c.left - c.width) < edge or abs(left + width - c.left) < edge then attach rectControl(window.control(i))
'end if
'else
'if (c.left > left-edge and c.left < left+width+edge) or (c.left+c.width > left-edge and c.left+c.width < left+width
+edge) or (left > c.left-edge and left+width < c.left+c.width+edge) then
'if abs(top - c.top - c.height) < edge or abs(top + height - c.top) < edge then attach rectControl(window.control(i))
'end if
'end if
'end if
'end if
'end if
'next

```

attach\_recursive(window)

End Sub

### SmartSplitter.attach\_recursive:

```
Sub attach_recursive(this_window as window)
    dim i, edge as integer
    dim okToAttach as boolean
    dim c as rectControl

    initializeSmartSplitter

    edge = 5 ' how close a control needs to be for it to auto-attach

    // doesn't handle recursively embedded controls
    for i = 0 to this_window.controlCount - 1
        if this_window.control(i) isA rectControl then
            c = rectControl(this_window.control(i))

            if parent = c.parent then
                okToAttach = true
                if parent isA tabPanel and tabPanelIndex <> c.tabPanelIndex then okToAttach = false
                if parent isA pagePanel and tabPanelIndex <> c.tabPanelIndex then okToAttach = false

            if okToAttach then
                if isVertical then
                    if (c.top > top-edge and c.top < top+height+edge) or (c.top+c.height > top-edge and c.top
                    +c.height < top+height+edge) or (top > c.top-edge and top+height < c.top+c.height+edge) then
                        if abs(left - c.left - c.width) < edge or abs(left + width - c.left) < edge then attach
                        rectControl(this_window.control(i))
                    end if
                else
                    if (c.left > left-edge and c.left < left+width+edge) or (c.left+c.width > left-edge and c.left+c.width
                    < left+width+edge) or (left > c.left-edge and left+width < c.left+c.width+edge) then
                        if abs(top - c.top - c.height) < edge or abs(top + height - c.top) < edge then attach
                        rectControl(this_window.control(i))
                    end if
                end if
            end if
        end if
    next

    if c isa window then
        attach_recursive(window(c))
    end if
end if
next

End Sub
```

### SmartSplitter.behave:

```
Sub behave()
    ' Call the behave method each time the parent window or control is resized.
    ' The behave method keeps the SmartSplitter locked to its MinLimit and MaxLimit
    ' if it was previously there and adjusts its proportions if stayProportional is true.

    initializeSmartSplitter
```

assignLimits

moveDistance 0, true

refreshDebugCanvas false

End Sub

### **SmartSplitter.controlCommonlyHasEmbeddedControls:**

Private Function controlCommonlyHasEmbeddedControls(c as rectControl) As Boolean

if c isA tabPanel or c isA pagePanel or c isA groupBox then return true

if c isA canvas or c isA rectangle then return true

if c isA editField or c isA listBox then return true

End Function

### **SmartSplitter.drawBar:**

Private Sub drawBar(x as integer, y as integer)

dim L as integer

dim light, medium, dark as color

#pragma disableBackgroundTasks

L = 5

if handleLarge then L = 9

if active and enabled or not initializedSmartSplitter then

light = colorLight

medium = colorMedium

dark = colorDark

else

light = colorLightDisabled

medium = colorMediumDisabled

dark = colorDarkDisabled

end if

if isVertical then

if handleLarge then

graphics.foreColor = light

graphics.DrawLine(x,y-L,x,y+L)

graphics.foreColor = medium

graphics.DrawLine(x-1,y-L,x-1,y+L) ' darker line 1 pixel to the left

graphics.DrawLine(x-1,y-L,x,y-L) ' nubs at ends of handle

graphics.DrawLine(x-1,y+L,x,y+L)

else

graphics.foreColor = dark

graphics.DrawLine(x,y-L,x,y+L)

end if

```

else 'is horizontal

if handleLarge then
    graphics.foreColor = light
    graphics.DrawLine(x-L,y,x+L,y)
    graphics.foreColor = medium
    graphics.DrawLine(x-L,y-1,x+L,y-1) ' darker line 1 pixel above
    graphics.DrawLine(x-L,y-1,x-L,y) ' nubs at ends of handle
    graphics.DrawLine(x+L,y-1,x+L,y)
else
    graphics.foreColor = dark
    graphics.DrawLine(x-L,y,x+L,y)
end if

end if

```

End Sub

### SmartSplitter.drawDimple:

```

Private Sub drawDimple(x as integer, y as integer)
    dim i, j as integer
    dim v as double
    dim c as color

    #pragma disableBackgroundTasks
    graphics.useOldRenderer = true

    if handleLarge then

        if active and enabled or not initializedSmartSplitter then

            for j = 1 to 5
                for i = 1 to 5
                    if dimplePixelValue(i,j) >= 0 then
                        graphics.foreColor = hsv(0,0,dimplePixelValue(i,j))
                        graphics.drawLine(x-3+i, y-3+j, x-3+i, y-3+j) ' used drawLine because pixel is unstable when
                        outside range of g
                    end if
                next
            next

        else

            for j = 1 to 5
                for i = 1 to 5
                    if dimplePixelValueDisabled(i,j) >= 0 then
                        graphics.foreColor = hsv(0,0,dimplePixelValueDisabled(i,j))
                        graphics.drawLine(x-3+i, y-3+j, x-3+i, y-3+j)
                    end if
                next
            next

        end if

    end if

```

else ' small handle

if not active and enabled or not initializedSmartSplitter then v = 0.2

if backgroundDark then ' darken dimples

graphics.foreColor = hsv(0,0,.68+v)

else

graphics.foreColor = hsv(0,0,.80+v)

end if

graphics.fillRect(x-1, y-1, 3, 3)

graphics.foreColor = hsv(0,0,.44+v)

graphics.drawLine(x-1, y, x, y-1)

graphics.foreColor = hsv(0,0,.6+v)

graphics.drawLine(x, y+1, x+1, y)

graphics.foreColor = hsv(0,0,.3+v)

graphics.drawLine(x, y, x, y)

end if

End Sub

### **SmartSplitter.drawHandle:**

Private Sub drawHandle()

dim x, y, n as integer

dim spacing as double

if conceal then return

x = width / 2

y = height / 2

n = 0

if top < 0 then n = top

if DisableLiveDrag and mouseDownTriggered then

graphics.foreColor = colorMedium

graphics.clearRect 0+1,n+1,width-2,height-n-2

graphics.drawRect 0,n,width,height-n

else

'if not targetMacOS then graphics.clearRect 0,n,width,height-n

end if

if handleDimple then

spacing = 3

if handleLarge then spacing = 5

for n = spacing-handleCount\*spacing to handleCount\*spacing step spacing\*2

if isVertical then

drawDimple x, y-n

else

drawDimple x-n, y

end if

next

else

spacing = 1

if handleLarge then spacing = 1.5

for n = spacing-handleCount\*spacing to handleCount\*spacing step spacing\*2

if isVertical then

drawBar x+n, y

else

drawBar x, y+n

end if

next

end if

End Sub

### **SmartSplitter.initializeSmartSplitter:**

Private Sub initializeSmartSplitter()

dim i, j, k as integer

dim x as double

dim s as string ' 5 x 5 matrix of grey value 0-100; negative for clear

if initializedSmartSplitter or initializingSmartSplitter then return

initializingSmartSplitter = true

isVertical = (height > width)

maxLimitPropSet = (maxLimit <> 0)

minLimitPropSet = (minLimit <> 0)

maxLimitOffsetPropSet = (maxLimitOffset <> 0)

minLimitOffsetPropSet = (minLimitOffset <> 0)

if debug then setDebug true

if swapOrientation then isVertical = not isVertical

if isVertical then

positionAtOpen = left

windowSizeAtOpen = window.width

else

positionAtOpen = top

windowSizeAtOpen = window.height

end if

windowSizeLast = windowSizeAtOpen

useFocusRing = false

showWarningsDialogs = debugBuild

maxLimitOffset = max(0,maxLimitOffset)

minLimitOffset = max(0,minLimitOffset)

atMaxLimit = false

```

atMinLimit = false
doNotAttachEmbeddedControls = false
if not minLimitPropSet then minLimit = 9999

if targetCarbon then
    backgroundDark = (window.frame = 9 and not parent isA tabPanel) ' metal window but not inside tab panel
else
    backgroundDark = true ' PPC, Win32, & Linux windows are darker
end if

colorLight = hsv(0,0,.75)
colorLightDisabled = hsv(0,0,.87)
colorMedium = hsv(0,0,.62)
colorMediumDisabled = hsv(0,0,.82)
colorDark = hsv(0,0,.55)
colorDarkDisabled = hsv(0,0,.70)

if backgroundDark then ' darken bars
    colorLight = hsv(0,0,.58)
    colorLightDisabled = hsv(0,0,.63)
    colorMedium = hsv(0,0,.50)
    colorMediumDisabled = hsv(0,0,.55)
    colorDark = hsv(0,0,.42)
    colorDarkDisabled = hsv(0,0,.50)
end if

assignCursor

if isVertical then
    if not maxLimitPropSet then maxLimit = window.width - width
else
    if not maxLimitPropSet then maxLimit = window.height - height
end if

setHandleType HandleType
'if targetWin32 then setHandleType 4 ' always set Win32 to large dimple

'----- set dimple pixel values
x = 1
if not backgroundDark then x = 1.1 ' lighten dimples
s = "-1,49, 9,48,-1,"
s = s + "48, 2,22,29,56,"
s = s + "11,37,64,75,65,"
s = s + "57,64,100,100,83,"
s = s + "-1,75,96,84,-1"
for j = 1 to 5
    for i = 1 to 5
        k = k + 1
        dimplePixelValue(i,j) = val(nthField(s,",",k)) / 100 * x
    next
next
s = "-1,62,42,62,-1,"
s = s + "61,38,48,51,65,"
s = s + "44,57,71,76,72,"

```

```

s = s + "67,70,88,89,80,"
s = s + "-1,77,87,81,-1"
k = 0
for j = 1 to 5
    for i = 1 to 5
        k = k + 1
        dimplePixelValueDisabled(i,j) = val(nthField(s,",",k)) / 100 * x
    next
next

```

unAttachAllControls

Open ' call sub's Open Event

if ubound(attachedControl) < 0 then attachNearbyControls

assignLimits

```

proportionValue = (top - proportionMin)/(proportionMax - proportionMin)
if isVertical then proportionValue = (left - proportionMin)/(proportionMax - proportionMin)

```

proportionAtOpen = proportionValue

```

initializedSmartSplitter = true
initializingSmartSplitter = false

```

End Sub

### SmartSplitter.moveDistance:

Sub moveDistance(d as integer, fromBehave as boolean = false)

' Moves splitter the given distance.

' behave = true will keep splitter at limits if previously there

dim p as integer

initializeSmartSplitter

if minLimit = 9999 then minLimit = 0

p = top

if isVertical then p = left

'if liveDrag or not behave then

if fromBehave and stayProportional then d = proportionMin + (proportionMax - proportionMin)\*proportionValue - p

if (p + d < minLimit + minLimitOffset/2 or (atMinLimit and fromBehave)) and not StopAtMinOffset then

atMinLimit = true

d = minLimit - p ' snap to minLimit

else

atMinLimit = false

if p + d < minLimit + minLimitOffset then d = minLimit + minLimitOffset - p ' catch on snapToMinLimitOffset

end if



```

if (p + d > maxLimit - maxLimitOffset/2 or (atMaxLimit and fromBehave)) and not StopAtMaxOffset then
    atMaxLimit = true
    d = maxLimit - p ' snap to maxLimit
else
    atMaxLimit = false
    if p + d > maxLimit - maxLimitOffset then d = maxLimit - maxLimitOffset - p ' catch on
    snapToMaxLimitOffset
end if

if p + d < minLimit then d = minLimit - p ' no less than minLimit
if p + d > maxLimit then d = maxLimit - p ' no higher than maxLimit

'end if

proportionValue = (p - proportionMin + d)/(proportionMax - proportionMin)
proportionValue = max(0, proportionValue)
proportionValue = min(1, proportionValue)

if d <> 0 or fromBehave then

    if isVertical then
        left = left + d
    else
        top = top + d
    end if

    #if enableDebugCanvasCode then
        if mouseDownTriggered and debug then debugCanvas.clearP
    #endif

    if DisableLiveDrag then

        if fromBehave then adjustAttachedControls d, fromBehave

    else

        adjustAttachedControls d, fromBehave

        Moved d

    end if

end if

```

End Sub

### **SmartSplitter.position:**

Function position() As integer

```

initializeSmartSplitter

if isVertical then
    return left

```

```
else
    return top
end if
```

End Function

### **SmartSplitter.proportion:**

```
Function proportion() As double
    initializeSmartSplitter
    return proportionValue
End Function
```

### **SmartSplitter.refreshDebugCanvas:**

```
Private Sub refreshDebugCanvas(clearFirst as boolean = true)
```

```
    #if enableDebugCanvasCode then

        if debugCanvas <> nil then
            if clearFirst then debugCanvas.clearP
            debugCanvas.markSplitter me, minLimit, maxLimit, isVertical
            debugCanvas.visible = false
            debugCanvas.visible = true
        end if

    #endif
```

End Sub

### **SmartSplitter.resetPosition:**

```
Sub resetPosition()
```

```
    ' Moves splitter to its original position at creation.
```

```
    initializeSmartSplitter

    if stayProportional then
        setProportion proportionAtOpen
    else
        if isVertical then
            if lockRight then
                setPosition positionAtOpen + window.width - windowSizeAtOpen
            else
                setPosition positionAtOpen
            end if
        else
            if lockBottom then
                setPosition positionAtOpen + window.height - windowSizeAtOpen
            else
                setPosition positionAtOpen
            end if
        end if
    end if
```

End Sub

### **SmartSplitter.setAttachedControlMinSizeVisible:**

```
Sub setAttachedControlMinSizeVisible(c as rectControl, x as integer)
    ' setting to < 0 will have no effect since negative sized controls are
    ' hidden automatically in adjustAttachedControls
    dim i as integer

    for i = 0 to ubound(attachedControl)
        if c = attachedControl(i) then
            attachedControlMinSizeVisible(i) = x
            return
        end if
    next

    warningForControlNotAttached c, "setAttachedControlMinSizeVisible"

End Sub
```

### **SmartSplitter.setAttachedControlNormallyVisible:**

```
Sub setAttachedControlNormallyVisible(c as rectControl, b as boolean)
    ' Sets if control should normally be visible.
    dim i as integer

    for i = 0 to ubound(attachedControl)
        if c = attachedControl(i) then
            attachedControlNormallyVisible(i) = b
            return
        end if
    next

    warningForControlNotAttached c, "setAttachedControlNormallyVisible"

End Sub
```

### **SmartSplitter.setAttachedControlShouldCenter:**

```
Sub setAttachedControlShouldCenter(c as rectControl, b as boolean)
    ' Sets if control should stay centered between min/max and splitter.
    ' Sets attachedControlShouldResize to False
    dim i as integer

    for i = 0 to ubound(attachedControl)
        if c = attachedControl(i) then
            attachedControlShouldCenter(i) = b
            if b then attachedControlShouldResize(i) = false
            assignLimits
            adjustAttachedControls 0
            return
        end if
    next

    warningForControlNotAttached c, "setAttachedControlShouldCenter"

End Sub
```

### **SmartSplitter.setAttachedControlShouldMove:**

Sub setAttachedControlShouldMove(c as rectControl, b as boolean)

' Sets if control should move with the splitter.

dim i as integer

for i = 0 to ubound(attachedControl)

if c = attachedControl(i) then

attachedControlShouldMove(i) = b

return

end if

next

warningForControlNotAttached c, "setAttachedControlShouldMove"

End Sub

### **SmartSplitter.setAttachedControlShouldResize:**

Sub setAttachedControlShouldResize(c as rectControl, b as boolean)

' Sets if control should resize with the splitter.

' Sets attachedControlShouldCenter to False

dim i as integer

for i = 0 to ubound(attachedControl)

if c = attachedControl(i) then

attachedControlShouldResize(i) = b

if b then attachedControlShouldCenter(i) = false

return

end if

next

warningForControlNotAttached c, "setAttachedControlShouldResize"

End Sub

### **SmartSplitter.setConceal:**

Sub setConceal(b as boolean)

' When set true, the splitter will not show its handle or mouseCursor

' and will ignore mouseDowns. (Move methods still work)

initializeSmartSplitter

conceal = b

refresh

End Sub

### **SmartSplitter.setDebug:**

Sub setDebug(b as boolean)

' When set true, additional information about the splitter and attached controls will be

' drawn to the CanvasSmartSplitterDebugger. (Requires that a canvas of

' class CanvasSmartSplitterDebugger be somewhere in the window. It will

' resize to the window. It should have top ControlOrder so as to not be

' hidden by other controls)

' Note that debugCanvas is always nil unless b=True and a CanvasSmartSplitterDebugger  
' is found. So no extra memory is consumed.

' If you wish to get rid of CanvasSmartSplitterDebugger completely, change the constant  
' "enableDebugCanvasCode" within this class from "True" to "False" and delete the  
' "debugCanvas" Property.

#if enableDebugCanvasCode then

dim i as integer

initializeSmartSplitter

debug = b

if b then 'look for a CanvasSmartSplitterDebugger class in the window

for i = 0 to window.controlCount-1

if window.control(i) isA CanvasSmartSplitterDebugger then

debugCanvas = CanvasSmartSplitterDebugger(window.control(i)) ' connect to it for debugging  
purposes

debugCanvas.debug = true

exit

end if

next

if debugCanvas = nil then

msgBox "SmartSplitter '"+name+"' needs a CanvasSmartSplitterDebugger to be present in the window for  
debug mode to work."

debug = false

end if

else

if debugCanvas <> nil then

debugCanvas.debug = false

refreshDebugCanvas

end if

debugCanvas = nil ' delete canvas

end if

refreshDebugCanvas

#else

debug = false

if b then msgBox "SmartSplitter '"+name+"' enableDebugCanvasCode is set to False so it cannot connect to  
CanvasSmartSplitterDebugger."

#endif

End Sub

### **SmartSplitter.setHandleType:**

Sub setHandleType(x as integer)

' Sets type of handle to be painted.

' -1 = No Handle; 0 = Default Handle;

- ' 1 = Small Bar Handle;
- ' 2 = Large Bar Handle;
- ' 3 = Small Dimple Handle;
- ' 4 = Large Dimple Handle

initializeSmartSplitter

select case x

case 0 ' auto-set

handleDimple = (window.frame = 9 and not parent isA tabPanel) ' metal window but not inside tab panel  
handleLarge = false

if isVertical then

if handleDimple then

handleCount = 3

if width > 5 and height > 20 then

handleLarge = true

handleCount = 1

end if

else

handleCount = 3

if width > 12 and height > 120 then

handleLarge = true

handleCount = 2

else

if width < 10 then handleCount = 2

end if

end if

else

if handleDimple then

handleCount = 3

if height > 5 and width > 20 then

handleLarge = true

handleCount = 1

end if

else

handleCount = 3

if height > 12 and width > 120 then

handleLarge = true

handleCount = 2

else

if height < 10 then handleCount = 2

end if

end if

end if

case 1 ' Small Bar

handleDimple = false

handleLarge = false

handleCount = 3

case 2 ' Large Bar

handleDimple = false

```

        handleLarge = true
        handleCount = 2
    case 3 ' Small Dimple
        handleDimple = true
        handleLarge = false
        handleCount = 3
    case 4 ' Large Dimple
        handleDimple = true
        handleLarge = true
        handleCount = 1
    else ' no handle
        handleCount = 0
    end select

```

```

refresh

```

```

End Sub

```

### **SmartSplitter.setMaxAttachedControl:**

```

Sub setMaxAttachedControl(c as rectControl, onlyIfBetterThanCurrent as boolean = false)

```

```

    ' Sets the maximum the splitter can be moved to always be just to the left or
    ' above the passed control. As controls are attached, the MinAttachedControl
    ' and MaxAttachedControl furthest from the splitter is selected.
    ' (This control is not required to be an attached control, but it would be strange if it weren't)

```

```

    initializeSmartSplitter

```

```

    if onlyIfBetterThanCurrent then

```

```

        if maxAttachedControl = nil then maxAttachedControl = c

```

```

        if isVertical then

```

```

            if c.left+c.width > maxAttachedControl.left+maxAttachedControl.width then maxAttachedControl = c

```

```

        else

```

```

            if c.top+c.height > maxAttachedControl.top+maxAttachedControl.height then maxAttachedControl = c

```

```

        end if

```

```

    else

```

```

        maxAttachedControl = c

```

```

    end if

```

```

    refreshDebugCanvas

```

```

End Sub

```

### **SmartSplitter.setMaxLimit:**

```

Sub setMaxLimit(x as integer)

```

```

    ' Sets the maximum the splitter can be moved. This limit will adjust with
    ' window resize. If wish it to stay fixed, set it in the IDE "Properties" window.

```

```

    initializeSmartSplitter

```

```

    maxLimit = x

```

```

    maxAttachedControl = nil

```

```

    refreshDebugCanvas

```

End Sub

### **SmartSplitter.setMaxLimitOffset:**

Sub setMaxLimitOffset(c as rectControl)

    initializeSmartSplitter  
    assignLimits

    if isVertical then  
        maxLimitOffset = maxLimit - c.left + width  
    else  
        maxLimitOffset = maxLimit - c.top + height  
    end if

    refreshDebugCanvas

End Sub

### **SmartSplitter.setMaxLimitOffset:**

Sub setMaxLimitOffset(x as integer)

    ' Sets the offset distance from the MaxLimit that the splitter will stop  
    ' at before snapping to the MaxLimit. Send <= 0 for no offset snap.

    initializeSmartSplitter  
    maxLimitOffset = max(x,0)  
    refreshDebugCanvas

End Sub

### **SmartSplitter.setMinAttachedControl:**

Sub setMinAttachedControl(c as rectControl, onlyIfBetterThanCurrent as boolean = false)

    ' Sets the maximum the splitter can be moved to always be just to the right or  
    ' below the passed control. As controls are attached, the MinAttachedControl  
    ' and MaxAttachedControl furthest from the splitter is selected.  
    ' (This control is not required to be an attached control, but it would be strange if it weren't)

    initializeSmartSplitter

    if onlyIfBetterThanCurrent then

        if minAttachedControl = nil then minAttachedControl = c

        if isVertical then  
            if c.left < minAttachedControl.left then minAttachedControl = c  
        else  
            if c.top < minAttachedControl.top then minAttachedControl = c  
        end if

    else  
        minAttachedControl = c  
    end if



```
refreshDebugCanvas
```

```
End Sub
```

### **SmartSplitter.setMinLimit:**

```
Sub setMinLimit(x as integer)
```

```
    ' Sets the minimum the splitter can be moved.
```

```
    initializeSmartSplitter
```

```
    minLimit = x
```

```
    minAttachedControl = nil
```

```
    refreshDebugCanvas
```

```
End Sub
```

### **SmartSplitter.setMinLimitOffset:**

```
Sub setMinLimitOffset(c as rectControl)
```

```
    initializeSmartSplitter
```

```
    assignLimits
```

```
    if isVertical then
```

```
        minLimitOffset = c.left + c.width - minLimit
```

```
    else
```

```
        minLimitOffset = c.top + c.height - minLimit
```

```
    end if
```

```
    refreshDebugCanvas
```

```
End Sub
```

### **SmartSplitter.setMinLimitOffset:**

```
Sub setMinLimitOffset(x as integer)
```

```
    ' Sets the offset distance from the MinLimit that the splitter will stop
```

```
    ' at before snapping to the MinLimit. Send <= 0 for no offset snap.
```

```
    initializeSmartSplitter
```

```
    minLimitOffset = max(x,0)
```

```
    refreshDebugCanvas
```

```
End Sub
```

### **SmartSplitter.setPosition:**

```
Sub setPosition(p as integer)
```

```
    ' Moves splitter to the given position. (From parent window Left or Top)
```

```
    dim x as integer
```

```
    initializeSmartSplitter
```

```
    if warningIfNonAttached("setPosition") then return
```

```
    x = position
```

```

if isVertical then
    moveDistance p - left
else
    moveDistance p - top
end if

if DisableLiveDrag then adjustAttachedControls position-x, true

refreshDebugCanvas

```

End Sub

### **SmartSplitter.setProportion:**

```

Sub setProportion(x as variant)
    ' Moves splitter to the given proportional position. (0 ≤ x ≤ 1)
    dim s as string
    dim b as boolean

    initializeSmartSplitter

    if warningIfNonAttached("setProportion") then return

    if showWarningsDialogs and varType(x) <> 5 and x <> 0 and x <> 1 then
        s = "The value type passed to the SmartSplitter ""+name+"" method 'setProportion' is not a single or double."
        s = s + endOfLine + endOfLine
        s = s + "Proportion must be referenced as a single or double – double is preferred as that is how it is referenced internally."
        msgBox s
        return
    end if

    proportionValue = x

    atMinLimit = false
    atMaxLimit = false

    b = stayProportional
    stayProportional = true

    behave

    stayProportional = b

    refreshDebugCanvas

End Sub

```

### **SmartSplitter.suggestMinSizeControlShouldBeVisible:**

```

Private Function suggestMinSizeControlShouldBeVisible(c as rectControl) As Integer
    ' Sets minimum size that control should remain visible.
    ' Setting to < 0 will have no effect since controls with a negative width
    ' or height are hidden automatically anyway to avoid display glitches.

    if c isA ListBox then return 20

```

```

if isVertical and (c isA checkBox or c isA radioButton) then return 18
if isVertical and c isA EditField then return 20
if isVertical and c isA PopupMenu then return 18
if c isA GroupBox or c isA PagePanel or c isA TabPanel then return 28

'if c isA SpriteSurface then return 20

'if c isA Rb3DSpace then return 20 'may require Qesa.dll on win32

'if c isA MoviePlayer then return 20 'may require QuickTime be installed

return 2

```

End Function

### **SmartSplitter.suggestOffsetSizeForControl:**

Private Function suggestOffsetSizeForControl(c as rectControl) As Integer

```

if c isA TabPanel then return 140

if c isA GroupBox or c isA PagePanel then return 100

if c isA ListBox then
    if isVertical then return 86
    if ListBox(c).scrollBarHorizontal then return 66
end if

if c isA EditField then
    if isVertical then
        if EditField(c).scrollBarVertical then return 80
        return 60
    end if
    if EditField(c).multiLine and EditField(c).scrollBarHorizontal then return 46
end if

if isVertical and (c isA checkBox or c isA radioButton) then return 60

if isVertical and (c isA PopupMenu or c isA StaticText) then return 40

return 30

```

End Function

### **SmartSplitter.suggestWhetherControlShouldResize:**

Private Function suggestWhetherControlShouldResize(c as rectControl) As Boolean

```

if c isA smartSplitter then
    if isVertical = (smartSplitter(c).height > smartSplitter(c).width) then return false 'height > width should mean it
    isVertical
end if

if isVertical then
    if c.lockLeft and c.lockRight then return true
else

```

```

    if c.lockTop and c.lockBottom then return true
end if

if c isA Separator or c isA Rectangle or c isA Placard then return true
if c isA ImageWell or c isA Canvas or c isA ListBox then return true

if c isA EditField then
    if isVertical or editField(c).multiLine then return true
end if

if c isA StaticText then
    if not isVertical and StaticText(c).multiLine then return true
end if

if isVertical and c isA PopupMenu then return true

if c isA GroupBox or c isA PagePanel or c isA TabPanel then return true

'if c isA SpriteSurface then return true

'if c isA Rb3DSpace then return true 'may require Quesa.dll on win32

'if c isA MoviePlayer then return true 'may require QuickTime be installed

return false

```

End Function

### **SmartSplitter.unAttach:**

Sub unAttach(c as rectControl, recursiveCall as boolean = false)

' Unattaches an attached control.

' also unAttaches embedded controls in c

dim i as integer

if c <> nil then

for i = 0 to ubound(attachedControl)

if c = attachedControl(i) then

attachedControl.Remove i

attachedControlBehind.Remove i

attachedControlMinSizeVisible.Remove i

attachedControlNormallyVisible.Remove i

attachedControlShouldCenter.Remove i

attachedControlShouldMove.Remove i

attachedControlShouldResize.Remove i

attachedControlSize.Remove i

if controlCommonlyHasEmbeddedControls(c) then

for i = 0 to window.controlCount-1 ' unAttach all controls inside this control

if rectControl(window.control(i)).parent = c then unAttach rectControl(window.control(i)), true

next

end if

return

end if

next

end if

if not recursiveCall then warningForControlNotAttached c, "unAttach"

End Sub

### **SmartSplitter.unAttachAllControls:**

Sub unAttachAllControls()

' Unattaches all attached controls.

dim i as integer

for i = 0 to ubound(attachedControl)

attachedControl(i) = nil

next

redim attachedControl(-1)

redim attachedControlBehind(-1)

redim attachedControlMinSizeVisible(-1)

redim attachedControlNormallyVisible(-1)

redim attachedControlShouldCenter(-1)

redim attachedControlShouldMove(-1)

redim attachedControlShouldResize(-1)

redim attachedControlSize(-1)

minAttachedControl = nil

maxAttachedControl = nil

End Sub

### **SmartSplitter.warningForControlNotAttached:**

Private Sub warningForControlNotAttached(c as rectControl, methodName as string)

dim s as string

if not showWarningsDialogs then return

if c = nil then

s = "The control passed to the SmartSplitter ""+name+"" method ""+methodName+"" is NIL."

s = s + endOfLine + endOfLine

s = s + "Don't go calling this method with nonexistent controls!"

msgBox s

else

s = "The control ""+c.name+"" passed to the SmartSplitter ""+name+"" method ""+methodName+"" has not been attached."

s = s + endOfLine + endOfLine

s = s + "Control must be attached BEFORE calling this method."

msgBox s

end if

End Sub

### **SmartSplitter.warningIfNonAttached:**

```

Private Function warningIfNonAttached(methodName as string) As Boolean
    dim s as string

    if ubound(attachedControl) < 0 then
        if showWarningsDialogs then
            s = "SmartSplitter '"+name+"' method '"+methodName+"' WARNING:"
            s = s + endOfLine + endOfLine
            s = s + "All controls should be attached BEFORE calling this method.  Call will be ignored."
            msgBox s
        end if
        return true
    end if
End Function

```

### **SmartSplitter.atMaxLimit:**

atMaxLimit as boolean

### **SmartSplitter.atMinLimit:**

atMinLimit as boolean

### **SmartSplitter.attachedControl(-1):**

attachedControl(-1) as rectControl

### **SmartSplitter.attachedControlBehind(-1):**

attachedControlBehind(-1) as boolean

### **SmartSplitter.attachedControlMinSizeVisible(-1):**

attachedControlMinSizeVisible(-1) as integer

### **SmartSplitter.attachedControlNormallyVisible(-1):**

attachedControlNormallyVisible(-1) as boolean

### **SmartSplitter.attachedControlShouldCenter(-1):**

attachedControlShouldCenter(-1) as boolean

### **SmartSplitter.attachedControlShouldMove(-1):**

attachedControlShouldMove(-1) as boolean

**SmartSplitter.attachedControlShouldResize(-1):**

attachedControlShouldResize(-1) as boolean

**SmartSplitter.attachedControlSize(-1):**

attachedControlSize(-1) as integer

**SmartSplitter.backgroundDark:**

backgroundDark as boolean

**SmartSplitter.colorDark:**

colorDark as color

**SmartSplitter.colorDarkDisabled:**

colorDarkDisabled as color

**SmartSplitter.colorLight:**

colorLight as color

**SmartSplitter.colorLightDisabled:**

colorLightDisabled as color

**SmartSplitter.colorMedium:**

colorMedium as color

**SmartSplitter.colorMediumDisabled:**

colorMediumDisabled as color

**SmartSplitter.conceal:**

conceal as boolean

**SmartSplitter.Debug:**

Debug as boolean

Private debugCanvas As CanvasSmartSplitterDebugger

**SmartSplitter.dimplePixelValue(5,5):**

dimplePixelValue(5,5) as double

**SmartSplitter.dimplePixelValueDisabled(5,5):**

dimplePixelValueDisabled(5,5) as double

**SmartSplitter.DisableLiveDrag:**

DisableLiveDrag as boolean

**SmartSplitter.doNotAttachEmbeddedControls:**

doNotAttachEmbeddedControls as boolean

**SmartSplitter.handleCount:**

handleCount as integer

**SmartSplitter.handleDimple:**

handleDimple as boolean

**SmartSplitter.handleLarge:**

handleLarge as boolean

**SmartSplitter.HandleType:**

HandleType as integer

**SmartSplitter.initializedSmartSplitter:**

initializedSmartSplitter as boolean



**SmartSplitter.initializingSmartSplitter:**

initializingSmartSplitter as boolean

**SmartSplitter.isVertical:**

isVertical as boolean

**SmartSplitter.maxAttachedControl:**

maxAttachedControl as rectControl

**SmartSplitter.MaxLimit:**

MaxLimit as integer

**SmartSplitter.MaxLimitOffset:**

MaxLimitOffset as integer

**SmartSplitter.maxLimitOffsetPropSet:**

maxLimitOffsetPropSet as boolean

**SmartSplitter.maxLimitPropSet:**

maxLimitPropSet as boolean

**SmartSplitter.minAttachedControl:**

minAttachedControl as rectControl

**SmartSplitter.MinLimit:**

MinLimit as integer

**SmartSplitter.MinLimitOffset:**

MinLimitOffset as integer

**SmartSplitter.minLimitOffsetPropSet:**

minLimitOffsetPropSet as boolean

**SmartSplitter.minLimitPropSet:**

minLimitPropSet as boolean

**SmartSplitter.mouseDownTriggered:**

mouseDownTriggered as boolean

**SmartSplitter.mouseDownX:**

mouseDownX as integer

**SmartSplitter.mouseDownY:**

mouseDownY as integer

**SmartSplitter.positionAtMouseDown:**

positionAtMouseDown as integer

**SmartSplitter.positionAtOpen:**

positionAtOpen as integer

**SmartSplitter.proportionAtOpen:**

proportionAtOpen as double

**SmartSplitter.proportionMax:**

proportionMax as integer

**SmartSplitter.proportionMin:**

proportionMin as integer

**SmartSplitter.proportionValue:**

proportionValue as double

**SmartSplitter.showWarningsDialogs:**

showWarningsDialogs as boolean

**SmartSplitter.StayProportional:**

StayProportional as boolean

**SmartSplitter.StopAtMaxOffset:**

StopAtMaxOffset as boolean

**SmartSplitter.StopAtMinOffset:**

StopAtMinOffset as boolean

**SmartSplitter.SwapOrientation:**

SwapOrientation as boolean

**SmartSplitter.windowSizeAtOpen:**

windowSizeAtOpen as integer

**SmartSplitter.windowSizeLast:**

windowSizeLast as integer

**SmartSplitter Note: How much is it ?**

How much is it ?

The SmartSplitter class is provided free for all to use. However, I do maintain ownership of SmartSplitter. You are free to redistribute the source code as long as this documentation goes with it.

There is no need to credit me in your program for using SmartSplitter unless you just want to.

If you use SmartSplitter in a project that generates actual profit (unlike any of mine;) or in a project for your work, you may feel compelled to offer compensation for the benefit it provided. Any monetary gift of thanks would be appreciated and would encourage continued development and support of SmartSplitter. To donate, go to <http://www.harryhooie.com/donate/>.

## SmartSplitter Note: Revision History

### Revision History

#### Version 1.3.2

- ClearRect removed from paint routine to fix background color under Windows XP
- Splitters set to StayProportional are now proportional to attached control extents rather than from min/maxLimits  
--- Thanks to Tomas Camin for letting me know about this ---
- Fixed CanvasSmartSplitterDebugger so that it now works in Mac OS X using RB2005 and later
- Masking CanvasSmartSplitterDebugger so that it looks better
- Fixed initialization issue happening splitter is referenced before Open Event

#### Version 1.3.1

- Uses REALbasic built in cursors in RB2005 and later
- Constant enableDebugCanvasCode can be changed to false (in IDE only) to completely skip all debug code so that CanvasSmartSplitterDebugger will no longer be required in your project

#### Version 1.3

- Fixed control smarts to handle embeded control resize change in REALbasic 2005
- Added logic to handle invisible attached controls  
(Use new subroutine setAttachedControlNormallyVisible to alert SmartSplitter if an attached control changes visibility)  
--- Thanks to Steve Roy for alerting me to this problem ---
- Using DisableLiveDrag looks MUCH better
- Fixed a Win32 "expanding" handle refresh problem

#### Version 1.2.2

- Changed main for-next loop in attachNearbyControls to loop to window.controlCount-1 - fixes a possible future OutOfBoundsException

#### Version 1.2.1

- When scanning for embedded controls only RectControls are attached - fixes a IllegalCastException
- Line controls are no longer attached - they are ignored since they don't work like other RectControls  
--- Thanks to Massimo Valle for alerting me to both these bugs ---

#### Version 1.2

- Set graphics.useOldRenderer = true in drawDimple method so that the dimples would be visible in Panther metal windows
- Added alternate more-Windows-looking cursors for win32 compiles
- Added EditField logic in suggestOffsetSizeForControl to take into account visibility of scrollbars

#### Version 1.1

- Darkened handle when on metal windows (but not inside tab panel)
- Proportion limited to between 0.0 and 1.0
- Commented out code checking if rectControl isA Rb3DSpace/MoviePlayer in suggestMinSizeControlShouldBeVisible and suggestWhetherControlShouldResize since this was requiring Quesa.dll/QuickTime on win32
- Added unAttach method for controls (why not?)

If you make any code improvements, you are welcome to send them to me at [smartsplitter@harryhooie.com](mailto:smartsplitter@harryhooie.com), and I will try to incorporate them in the next update so that everyone using SmartSplitter can benefit. I am particularly interested in a Win32 declare to draw a line/rect/region (but not a drag-region) to lessen the flickering - so if anyone has a declare routine coded up...

## SmartSplitter Note: Using SmartSplitter

### Using SmartSplitter

First add the SmartSplitter class to your project by dragging the "SmartSplitter" folder into your project window.

To add a SmartSplitter to one of your project windows, add a canvas to the window and change the canvas's Super property from "Canvas" to "SmartSplitter". (or drag the SmartSplitter class from Project Controls onto the window) Now resize it so that it covers the area between the controls you wish to be adjustable.

If the splitter's height is greater than its width, it will act as a vertical splitter and look for controls to the left and right. Otherwise it will act as a horizontal splitter looking for controls above and below. Controls with edges within 5 pixels of the edges of the splitter will automatically attach unless you command otherwise.

All controls you wish to adjust with the splitter should be position locked as they would need to be to resize correctly with their container window – even if the window is not going to be resizable.

### \*\*\* Behave

If your window is to be resizable, a call to each SmartSplitter's behave method needs to be added to the parent window's Resized event (or to the Resizing event if the window is set to LiveResize):

```
Sub Resized()  
    SmartSplitter1.behave  
    SmartSplitter2.behave  
End Sub
```

... and ...

```
Sub Resizing()  
    SmartSplitter1.behave  
    SmartSplitter2.behave  
End Sub
```

The behave method keeps the SmartSplitter locked to its MinLimit and MaxLimit if it was previously locked and adjusts its proportions if StayProportional is selected (see Setting SmartSplitter properties) was set to true.

End Class

## Class CanvasSmartSplitterDebugger

### **CanvasSmartSplitterDebugger.Open:**

```
Sub Open()  
    dim w, h as integer  
  
    lockLeft = true  
    lockTop = true  
    lockRight = true  
    lockBottom = true  
    left = 0  
    top = 0  
    width = window.width  
    height = window.height  
  
    boxHeight = 11
```

End Sub

### **CanvasSmartSplitterDebugger.Paint:**

```
Sub Paint(g As Graphics)  
    dim w, h as integer  
  
    w = maxXdrawn-minXdrawn  
    h = maxYdrawn-minYdrawn  
    if debug and p <> nil then g.DrawPicture p,minXdrawn,minYdrawn,w,h,minXdrawn,minYdrawn,w,h  
  
    if lastWidth <> width or lastHeight <> height then clearP  
  
    lastWidth = width  
    lastHeight = height
```

End Sub

### **CanvasSmartSplitterDebugger.clearP:**

```
Sub clearP()  
  
    if createP then return ' p still nil  
  
    p.graphics.clearRect 0,0,width,height  
  
    p.mask.graphics.foreColor = &cFFFFFF  
    p.mask.graphics.fillRect 0,0,width,height  
  
    minXdrawn = width  
    minYdrawn = height  
    maxXdrawn = 0  
    maxYdrawn = 0
```

End Sub

**CanvasSmartSplitterDebugger.createP:**

Private Function createP() As Boolean

```
if debug and p = nil then
    if targetMacOS or targetWin32 then
        if targetPPC then
            p = newPicture(1024,768,16) ' limit canvas so Mac OS 9 won't crash
        else
            p = newPicture(screen(0).width,screen(0).height,screen(0).depth)
        end if
    else
        p = newPicture(1024,768,screen(0).depth) ' limit canvas so Linux won't crash
    end if

    if p = nil then
        if not alreadyWarned then
            alreadyWarned = true
            msgbox "CanvasSmartSplitterDebugger can not be initialized. Not enough available memory."
        end if
    else
        p.graphics.penWidth = 2
        p.graphics.penHeight = 2
        p.graphics.textFont = "Geneva"
        p.graphics.textSize = 9
        p.graphics.bold = true
        if targetMacOS or targetWin32 then p.transparent = 1
    end if
end if

return (p = nil) ' return true if p = nil
```

End Function

**CanvasSmartSplitterDebugger.drawBox:**

Private Sub drawBox(c as rectControl)

```
drawRect c.left,c.top,c.width,c.height
```

```
if not c.visible then drawX c
```

End Sub

**CanvasSmartSplitterDebugger.drawBoxedString:**

Private Sub drawBoxedString(s as string, x as integer, y as integer, justification as integer = 0)

```
dim boxWidth as integer
```

```
dim originalColor as color
```

```
boxWidth = p.graphics.stringWidth(s)+2
```

```
if justification = 1 then x = x - boxWidth/2 'center justified
```

```
if justification = 2 then x = x - boxWidth 'right justified
```

```
p.graphics.fillRect x, y, boxWidth, boxHeight
```

```
p.mask.graphics.foreColor = &c222222
p.mask.graphics.fillRect x, y, boxWidth, boxHeight
```

```
originalColor = p.graphics.foreColor
p.graphics.foreColor = hsv(0,0,0.94)
p.graphics.drawString s, x + 1, y + 9
p.graphics.foreColor = originalColor
```

End Sub

### **CanvasSmartSplitterDebugger.drawLine:**

Private Sub drawLine(x1 as integer, y1 as integer, x2 as integer, y2 as integer)

```
p.graphics.drawLine x1, y1, x2, y2
p.mask.graphics.foreColor = &c444444
p.mask.graphics.drawLine x1, y1, x2, y2
```

End Sub

### **CanvasSmartSplitterDebugger.drawRect:**

Private Sub drawRect(x as integer, y as integer, w as integer, h as integer)

```
p.graphics.drawRect x, y, w, h
p.mask.graphics.foreColor = &c444444
p.mask.graphics.drawRect x, y, w, h
```

```
minXdrawn = min(minXdrawn,x)
minYdrawn = min(minYdrawn,y)
maxXdrawn = max(maxXdrawn,x+w)
maxYdrawn = max(maxYdrawn,y+h)
```

End Sub

### **CanvasSmartSplitterDebugger.drawX:**

Private Sub drawX(c as rectControl)

```
drawLine c.left,c.top,c.left+c.width-2,c.top+c.height-2
drawLine c.left,c.top+c.height-2,c.left+c.width-2,c.top
```

End Sub

### **CanvasSmartSplitterDebugger.markControl:**

Sub markControl(c as rectControl)

```
if p = nil then return
```

```
p.graphics.foreColor = rgb(200,0,0)
```

```
drawBox c
```

```
'drawBoxedString str(c.width)+"x"+str(c.height), c.left+c.width/2, c.top+c.height/2-boxHeight/2, 1
```

End Sub



### CanvasSmartSplitterDebugger.markSplitter:

Sub markSplitter(c as SmartSplitter, minLimit as integer, maxLimit as integer, isVertical as boolean)

dim maxOffsetPos, minOffsetPos as integer

if p = nil then return

p.graphics.foreColor = rgb(0,150,0)

drawBox c

if isVertical then

p.graphics.foreColor = rgb(0,0,150)

minOffsetPos = minLimit+c.minLimitOffset

drawLine minOffsetPos,c.top,minOffsetPos,c.top+c.height-2

drawBoxedString "minLOffset="+str(c.minLimitOffset), minOffsetPos, c.top+c.height/2-boxHeight, 0

maxOffsetPos = maxLimit-c.maxLimitOffset

drawLine maxOffsetPos,c.top,maxOffsetPos,c.top+c.height-2

drawBoxedString "maxLOffset="+str(c.maxLimitOffset), maxOffsetPos, c.top+c.height/2, 2

p.graphics.foreColor = rgb(0,150,0)

drawRect minLimit,c.top,maxLimit-minLimit,c.height

drawLine minLimit+20,c.top,maxLimit-22,c.top+c.height-2

drawBoxedString "minLimit="+str(minLimit), minLimit, c.top

drawBoxedString "maxLimit="+str(maxLimit), maxLimit, c.top+c.height-boxHeight, 2

else

p.graphics.foreColor = rgb(0,0,150)

minOffsetPos = minLimit+c.minLimitOffset

drawLine c.left,minOffsetPos,c.left+c.width-2,minOffsetPos

drawBoxedString "minLOffset="+str(c.minLimitOffset), c.left+c.width/2, minOffsetPos, 1

maxOffsetPos = maxLimit-c.maxLimitOffset

drawLine c.left,maxOffsetPos,c.left+c.width-2,maxOffsetPos

drawBoxedString "maxLOffset="+str(c.maxLimitOffset), c.left+c.width/2, maxOffsetPos-boxHeight+2, 1

p.graphics.foreColor = rgb(0,150,0)

drawRect c.left,minLimit,c.width,maxLimit-minLimit

drawLine c.left+20,minLimit,c.left+c.width-22,maxLimit-2

drawBoxedString "minLimit="+str(minLimit), c.left, minLimit

drawBoxedString "maxLimit="+str(maxLimit), c.left+c.width, maxLimit-boxHeight, 2

end if

End Sub

### CanvasSmartSplitterDebugger.alreadyWarned:

alreadyWarned As boolean

**CanvasSmartSplitterDebugger.boxHeight:**  
boxHeight As integer

**CanvasSmartSplitterDebugger.debug:**  
debug As boolean

**CanvasSmartSplitterDebugger.lastHeight:**  
lastHeight As integer

**CanvasSmartSplitterDebugger.lastWidth:**  
lastWidth As integer

**CanvasSmartSplitterDebugger.maxXdrawn:**  
maxXdrawn As integer

**CanvasSmartSplitterDebugger.maxYdrawn:**  
maxYdrawn As integer

**CanvasSmartSplitterDebugger.minXdrawn:**  
minXdrawn As integer

**CanvasSmartSplitterDebugger.minYdrawn:**  
minYdrawn As integer

**CanvasSmartSplitterDebugger.p:**  
p As picture

**CanvasSmartSplitterDebugger.windowNotResizing:**  
windowNotResizing As boolean

End Class

## **Class Container Scroller**

Inherits ContainerControl

### **Container\_Scroller.Resized:**

```
Sub Resized()  
    update  
End Sub
```

### **Container\_Scroller.Resizing:**

```
Sub Resizing()  
    //Update  
End Sub
```

### **Container\_Scroller.add:**

```
Sub add(input as containerControl)  
    dim temp_height as integer = 14  
    if UBound(controls) > -1 then  
        temp_height = temp_height + controls(0).Height  
    end  
    for i as integer = 0 to UBound(controls)  
        temp_height = temp_height + 12  
    next  
    for i as integer = 1 to UBound(controls)  
        temp_height = temp_height + controls(i).Height  
    next  
    controls.Append input  
    input.EmbedWithin(container_canvas, 20, temp_height)  
    temp_height = temp_height + 20 + input.Height  
    temp_height = temp_height + 20  
  
    dim temp_width as integer = 0  
  
    for i as integer = 0 to UBound(controls)  
        if temp_width < controls(i).Width + 20 + 20 then //left padding, right padding  
            temp_width = controls(i).Width + 20 + 20  
        end  
    next  
  
    calculated_width = temp_width + 20 //just because  
    calculated_height = temp_height  
  
    update  
End Sub
```

### **Container\_Scroller.Constructor:**

```
Sub Constructor()  
    canvas_ref = container_canvas  
End Sub
```

### **Container\_Scroller.Destructor:**

```
Sub Destructor()  
    kill  
End Sub
```

### **Container\_Scroller.increase\_dim:**

```
Sub increase_dim(height_add as integer = 0, width_add as integer = 0)  
    calculated_height = calculated_height + height_add  
    calculated_width = calculated_width + width_add  
    update  
End Sub
```

### **Container\_Scroller.kill:**

```
Sub kill()  
    while UBound(controls) > -1  
        controls(0).Close  
        controls.Remove(0)  
    wend  
  
    self.Close  
End Sub
```

### **Container\_Scroller.update:**

```
Sub update()  
    if calculated_height > self.Height - 16 then  
        canvas_scroll_vert.Maximum = calculated_height - self.Height - 16  
        canvas_scroll_vert.Minimum = 0  
        canvas_scroll_vert.Value = 0  
        canvas_scroll_vert.LineStep = (calculated_height - self.Height - 16)/100  
        if canvas_scroll_vert.LineStep = 0 then canvas_scroll_vert.LineStep = 1  
        canvas_scroll_vert.PageStep = (calculated_height - self.Height - 16)/5  
        if canvas_scroll_vert.PageStep = 0 then canvas_scroll_vert.PageStep = 1  
        previous_scroll_value_vert = 0  
        canvas_scroll_vert.Enabled = true  
    else  
        canvas_scroll_vert.Maximum = 0  
        canvas_scroll_vert.Minimum = 0  
        canvas_scroll_vert.Value = 0  
        previous_scroll_value_vert = 0  
        canvas_scroll_vert.Enabled = false  
    end  
  
    if calculated_width > self.width - 16 then  
        canvas_scroll_horiz.Maximum = calculated_width - self.width - 16  
        canvas_scroll_horiz.Minimum = 0  
        canvas_scroll_horiz.Value = 0  
        canvas_scroll_horiz.LineStep = (calculated_width - self.width - 16)/100  
        if canvas_scroll_horiz.LineStep = 0 then canvas_scroll_horiz.LineStep = 1  
        canvas_scroll_horiz.PageStep = (calculated_width - self.width - 16)/5  
        if canvas_scroll_horiz.PageStep = 0 then canvas_scroll_horiz.PageStep = 1  
        previous_scroll_value_horiz = 0
```

```

        canvas_scroll_horiz.Enabled = true
    else
        canvas_scroll_horiz.Maximum = 0
        canvas_scroll_horiz.Minimum = 0
        canvas_scroll_horiz.Value = 0
        previous_scroll_value_horiz = 0
        canvas_scroll_horiz.Enabled = false
    end

```

```

    me.Refresh(true)
End Sub
calculated_height As Integer

```

```

calculated_width As Integer

```

```

canvas_ref As canvas

```

```

controls(-1) As containerControl

```

```

previous_scroll_value_horiz As Integer

```

```

previous_scroll_value_vert As Integer

```

```

scroll_vals(-1) As double

```

## Container\_Scroller Note: spacing

spacing

14 pixels between top and first control  
 12 pixels between two controls  
 20 pixels to the left, 20 pixels to the right  
 20 pixels to the bottom

## Container\_Scroller Control canvas\_scroll\_vert:

```

Sub ValueChanged()
    if me.Enabled then
        'dim temp_change as integer = previous_scroll_value_vert - me.Value
        'dim temp_range as integer = me.Maximum - me.Minimum
        'dim temp_dev as double = abs(temp_change/temp_range)
        container_canvas.Scroll(0, previous_scroll_value_vert - me.Value, container_canvas.width,
        container_canvas.height)
        container_canvas.Refresh
        previous_scroll_value_vert = me.Value
    end
End Sub

```

## Container\_Scroller Control canvas\_scroll\_horiz:

```

Sub ValueChanged()
    if me.Enabled then
        container_canvas.Scroll(previous_scroll_value_horiz - me.Value, 0, container_canvas.width,
        container_canvas.height)
        previous_scroll_value_horiz = me.Value
        container_canvas.Refresh
    end
End Sub

```

```
end
End Sub
End Class
```

## **Module H1**

### **H1.CE\_replace:**

```
Function CE_replace(tags_to_replace() as string) As meta()
    dim temp_meta(-1) as meta
    dim internalize as Boolean = false

    Dim d as New MessageDialog //declare the MessageDialog object
    Dim b as MessageDialogButton //for handling the result
    d.icon=MessageDialog.GraphicNote //display app icon
    d.ActionButton.Caption="Replace"
    d.CancelButton.Visible= True //show the Cancel button
    d.AlternateActionButton.Visible= True //show the "Replace and Internalize" button
    d.AlternateActionButton.Caption="Replace and Internalize"
    d.Message="Select what to do for indexed tag replacement"
    d.Explanation="Replace will let you select a maps folder to get the replacement tags from." _
    + " Replace and internalize will do the same while also internalizing any bitmap and sound data." _
    + " Cancel will only cancel the replacement step and continue with the extraction as normal."
    b=d.ShowModal //display the dialog
    Select Case b //determine which button was pressed.
    Case d.ActionButton
        //user pressed Replace, continue

    Case d.AlternateActionButton
        //user pressed Replace and Internalize
        internalize = true

    Case d.CancelButton
        //user pressed Cancel
        return temp_meta
    End select

    Dim dlg as New SelectFolderDialog
    Dim f as FolderItem
    dlg.ActionButtonCaption="Select"
    dlg.Title="Maps Selection"
    dlg.PromptText="Select the folder with maps containing tags to replace the indexed tags"
    f=dlg.ShowModal()
    if f = nil then return temp_meta

    //something
    dim finished as Boolean = false
    dim t as new CE_tag_replacement_thread
    t.init(f, internalize, tags_to_replace)
    t.run
    while not t.finished
```

```

    //wait out the process
    //dim i as integer = 0
wend
for i as integer = 0 to t.replaced metas.Ubound
    temp_meta.Append t.replaced metas(i)
next
redim tags_to_replace(-1)
for i as integer = 0 to t.tags_to_replace.Ubound
    tags_to_replace.append t.tags_to_replace(i)
next

if tags_to_replace.Ubound > -1 then
    //did not replace all the tags. output an error and see if they want to try again

    dim tags_missed_str as string = ""
    for i as integer = 0 to tags_to_replace.ubound
        tags_missed_str = tags_missed_str + tags_to_replace(i).mid(1,4).reverse + ": "
        tags_missed_str = tags_missed_str + tags_to_replace(i).mid(6)
        if i < tags_to_replace.ubound then
            tags_missed_str = tags_missed_str + EndOfLine
        end
    next

    Dim d2 as New MessageDialog //declare the MessageDialog object
    Dim b2 as MessageDialogButton //for handling the result
    d2.icon=MessageDialog.GraphicCaution //display warning icon
    d2.ActionButton.Caption="Continue"
    d2.CancelButton.Visible= True //show the Cancel button
    d2.Message="Warning: the following tags were not replaced:" + EndOfLine + tags_missed_str
    d2.Explanation="Would you like to continue replacing or cancel and return to the tag extraction?"

    b2=d2.ShowModal //display the dialog
    Select Case b //determine which button was pressed.
    Case d2.ActionButton
        //user pressed Save
        dim more_meta() as h1.meta = CE_replace(tags_to_replace)
        for i as integer = 0 to UBound(more_meta)
            temp_meta.Append more_meta(i)
        next
    Case d2.CancelButton
        //user pressed Cancel
    End select
end

return temp_meta
End Function

```

## H1.internalizeTag:

```

Function internalizeTag(tag as meta, data_f as folderItem) As boolean
    if tag.CE_flag = 1 then return false//quit it
    if not tag.expanded then return false

    //theoretically this should work
    dim return_bool as Boolean = true

```

```

select case tag.class1
case "!dns"
    //extract the raw data for sounds if external
    dim s as new snd_class_EX(tag.data.bin, tag.offset, tag.magic)
    s.read
    for i as integer = 0 to UBound(s.pitch)
        for j as integer = 0 to UBound(s.pitch(i).permutations)
            if not s.pitch(i).permutations(j).internal then
                dim temp_raw as new Meta_Raw
                dim raw_writer as new BinaryStream( temp_raw.bin )
                temp_raw.bin.size = s.pitch(i).permutations(j).size
                temp_raw.size = s.pitch(i).permutations(j).size
                dim br as BinaryStream = data_f.OpenAsBinaryFile
                br.LittleEndian = true
                br.Position = s.pitch(i).permutations(j).offset
                for k as integer = 1 to s.pitch(i).permutations(j).size
                    raw_writer.WriteByte(br.ReadByte)
                next
                temp_raw.str_info = "snd!:pitch:" + str(i) + ":permutation:" + str(j)
                dim temp_bool as boolean = s.update_offset(i, j, s.pitch(i).permutations(j).offset, true)
                if temp_bool then
                    tag.raw_data.append temp_raw
                else
                    return_bool = false
                end
                br.Close
            end
        next
    next
case "mtib"
    //extract the raw data for bitmaps if internalized
    dim b as new bitmap_class_EX(tag.data.bin, tag.offset, tag.magic)
    b.read
    for i as integer = 0 to UBound(b.image_info)
        if (bitand(b.image_info(i).flags, &h100) = &h100) then
            //external bitmap data
            dim temp_raw as new Meta_Raw
            dim raw_writer as new BinaryStream( temp_raw.bin )
            temp_raw.bin.size = b.image_info(i).size
            temp_raw.size = b.image_info(i).size
            dim br as BinaryStream = data_f.OpenAsBinaryFile
            br.LittleEndian = true
            br.position = b.image_info(i).offset
            for j as integer = 1 to b.image_info(i).size
                raw_writer.writebyte(br.readbyte)
            next
            temp_raw.str_info = "bitm:image_info:" + str(i)
            dim temp_bool as boolean = b.update_offset(i, b.image_info(i).offset, true)
            if temp_bool then
                tag.raw_data.append temp_raw
            else
                return_bool = false
            end
        end
    next
end

```



```

        br.Close
    end
next
end select

return return_bool
End Function

```

## H1.load\_meta:

```

Function load_meta(meta_f as folderItem, xml_f as folderItem, data_f as folderItem = nil) As meta
    if meta_f = nil then return nil
    if xml_f = nil then return nil

    dim temp_meta as new meta
    //preliminary data entry
    temp_meta.CE_flag = 0
    temp_meta.expanded = true
    temp_meta.nameoffset = -1
    temp_meta.tagID = &hFFFFFFF
    temp_meta.zero = 0
    temp_meta.data = new Meta_Expanded_Data

    dim br as BinaryStream = meta_f.openasbinaryfile
    br.LittleEndian = true
    br.Position = 0
    dim bw as new BinaryStream(temp_meta.data.bin)
    bw.LittleEndian = true
    bw.Position = 0

    dim temp_size as integer = 0
    while not br.EOF
        bw.writebyte(br.readbyte)
        temp_size = temp_size + 1
    wend
    br.close
    bw.close
    temp_meta.data.size = temp_size
    temp_meta.metasize = temp_size

    dim xdoc as new XmlDocument
    xdoc.LoadXml(xml_f)

    //map should be first
    dim map_node as XmlNode
    map_node = xdoc.DocumentElement.Child(0)
    dim temp_map_name as string = map_node.getAttribute("name")
    dim temp_map_version as string = map_node.getAttribute("version")
    dim temp_magic as string = map_node.getAttribute("magic")
    temp_magic = temp_magic.replaceAll("0x","&h")
    temp_meta.map_name = temp_map_name
    temp_meta.map_version = val(temp_map_version)
    temp_meta.magic = val(temp_magic)

    //then tag

```

```

dim tag_node as XmlNode
tag_node = xdoc.DocumentElement.Child(1)
dim temp_class1 as string = tag_node.getAttribute("class1")
dim temp_class2 as string = tag_node.getAttribute("class2")
dim temp_class3 as string = tag_node.getAttribute("class3")
dim temp_tagoffset as string = tag_node.getAttribute("tagoffset")
temp_tagoffset = temp_tagoffset.replaceAll("0x","&h")
temp_meta.class1 = temp_class1
temp_meta.class2 = temp_class2
temp_meta.class3 = temp_class3
temp_meta.offset = val(temp_tagoffset)

//then filename
dim name_node as XmlNode
name_node = xdoc.DocumentElement.Child(2)
dim temp_tagname as string = name_node.getAttribute("name")
temp_meta.name = temp_tagname

//reflexives
dim reflexive_list as XmlNodeList
reflexive_list = xdoc.DocumentElement.Xql("//Reflexive")
for i as integer = 0 to reflexive_list.length - 1
    dim temp_node as new XmlNode
    temp_node = reflexive_list.item(i)
    dim location as string = temp_node.getAttribute("location")
    location = location.replaceAll("0x","&h")
    dim chunkcount as string = temp_node.getAttribute("chunkcount")
    dim translation as string = temp_node.getAttribute("translation")
    translation = translation.replaceAll("0x","&h")
    dim temp_reflexive as new reflexive
    temp_reflexive.offset = val(location)
    temp_reflexive.count = val(chunkcount)
    temp_reflexive.translation = val(translation)
    temp_meta.data.reflexives.append temp_reflexive
next

//dependencies
dim dependency_list as XmlNodeList
dependency_list = xdoc.DocumentElement.Xql("//Dependency")
for i as integer = 0 to dependency_list.Length - 1
    dim temp_node as new XmlNode
    temp_node = dependency_list.item(i)
    dim location as string = temp_node.getAttribute("location")
    location = location.replaceAll("0x","&h")
    dim tagclass as string = temp_node.getAttribute("tagclass")
    dim filename as string = temp_node.getAttribute("filename")
    dim temp_ref as new tag_reference
    temp_ref.offset = val(location)
    temp_ref.tag_class = tagclass
    temp_ref.tag_name = filename
    temp_meta.data.dependencies.append temp_ref
next

//lonelDs

```

```

dim loneID_list as XmlNodeList
loneID_list = xdoc.DocumentElement.Xql("//LoneID")
for i as integer = 0 to loneID_list.Length - 1
    dim temp_node as new XmlNode
    temp_node = loneID_list.item(i)
    dim location as string = temp_node.getAttribute("location")
    location = location.replaceAll("0x","&h")
    dim tagclass as string = temp_node.getAttribute("tagclass")
    dim filename as string = temp_node.getAttribute("filename")
    dim temp_ref as new tag_reference
    temp_ref.offset = val(location)
    temp_ref.tag_class = tagclass
    temp_ref.tag_name = filename
    temp_meta.data.loneIDs.append temp_ref
next

if data_f <> nil and data_f.Directory then
    //time to import all of the raw data as well
    for i as integer = 1 to data_f.Count
        dim raw_f as FolderItem = data_f.Item(i)
        if raw_f.Type = H1_Filetypes.EschatonBINFile.Name then
            dim temp_raw as new Meta_Raw
            temp_raw.offset = -1
            temp_raw.str_info = raw_f.Name.ReplaceAll("|",":")
            temp_raw.str_info = temp_raw.str_info.ReplaceAll(".bin","")
            dim raw_br as BinaryStream = raw_f.OpenAsBinaryFile
            raw_br.LittleEndian = true
            raw_br.Position = 0
            dim raw_bw as new BinaryStream(temp_raw.bin)
            raw_bw.LittleEndian = true
            raw_bw.Position = 0
            dim raw_size as integer = 0
            while not raw_br.EOF
                raw_bw.WriteByte(raw_br.ReadByte)
                raw_size = raw_size + 1
            wend
            raw_br.close
            raw_bw.close
            temp_raw.size = raw_size
            temp_meta.raw_data.Append temp_raw
        end
    next
end

return temp_meta
End Function

```

## H1.SortByOffset:

```

Function SortByOffset(Extends f() As meta) As integer()
    //doesn't handle arrays with Nil elements
    Dim v() As integer
    Dim k As Integer
    Do Until (k > UBound(f))
        v.Append(f(k).offset) // get the property value here
    Loop

```

```

    k = k + 1
Loop
dim orig(-1) as integer
for i as integer = 0 to UBound(f)
    orig.append i
next
v.SortWith(f, orig) // here REALbasic does the sort for me

Return orig
End Function

```

## **H1 Note: internal version numbers and release numbers**

internal version numbers and release numbers  
 anyone who finds this might wonder why I may refer to it as v4 code.  
 the file might even be called eschatonv4.rbp

the version number refers to the backend upgrade cycle. I've rewritten how eschaton reads and interprets meta editing several times. Always to increase capability

v0: HME 0.2 (halo mac editor)  
 single window based, hardcoded editing, no understanding of reflexives

v1: Eschaton 0.3  
 replaced editing windows to allow for editing several bitmasks, floats and the like as well as having reflexives but not using them correctly. major update was to use plugins

v2: Eschaton 0.4  
 editing windows received minor updates and now properly interpreted bitmasks  
 major update was to write code to handle reflexive chunks correctly

v3: Eschaton 0.5, 0.6  
 multiple instances of windows and a more refined method of controlling reflexives by use of the folder system (yuck). Also, allowed for recursive reflexives.

v4: Eschaton 0.7  
 single instance of a window (might end up writing multiplicity in)  
 upgraded backend to use extracted metas (hopefully to enable simple rebuilding)  
 much better editor interface

Now obviously the release numbers come off as weird. They involve two factors:  
 Which step number I consider the application to be on the way to a finished product  
 Something close to how far complete the program is capability wise.

0.1:  
 Being able to actually open and read a halo map file. Got source from cloud, but it didn't work. once the error was found and corrected, I consider that to be 0.1

HME 0.2:  
 working from 0.1, developed a quick little hard coded editor for metas I cared about. big deal when mac people had to hex edit for that sort of capability

Eschaton 0.3:  
 switched to plugins, lots of editing capability, dependency swapper (only reason I used HMT anyways)

0.4:

improved d-swapper, upgraded to reflexive level editing, simplified scnr editor, model viewing

0.5:

edit multiple maps at once, recursive reflexive editing

0.6:

3D scnr editor

0.7:

hopefully: rebuilding, bitmap-sound-model view/extract/inject

Global Enum BitmFormat As Integer

A8 = &h0

Y8 = &h1

AY8 = &h2

A8Y8 = &h3

R5G6B5 = &h6

A1R5G5B5 = &h8

A4R4G4B4 = &h9

X8R8G8B8 = &hA

A8R8G8B8 = &hB

DXT1 = &hE

DXT2\_3 = &hF

DXT4\_5 = &h10

P8 = &h11

End Enum

Global Enum DDSEnum As Integer

DDSD\_CAPS = &h1

DDSD\_HEIGHT = &h2

DDSD\_WIDTH = &h4

DDSD\_PITCH = &h8

DDSD\_PIXELFORMAT = &h1000

DDSD\_MIPMAPCOUNT = &h20000

DDSD\_LINEARSIZE = &h80000

DDSD\_DEPTH = &h800000

DDPF\_ALPHAPIXELS = &h1

DDPF\_FOURCC = &h4

DDPF\_RGB = &h40

DDSCAPS\_COMPLEX = &h8

DDSCAPS\_TEXTURE = &h1000

DDSCAPS\_MIPMAP = &h400000

DDSCAPS2\_CUBEMAP = &h200

DDSCAPS2\_CUBEMAP\_POSITIVEX = &h400

DDSCAPS2\_CUBEMAP\_NEGATIVEX = &h800

DDSCAPS2\_CUBEMAP\_POSITIVEY = &h1000

DDSCAPS2\_CUBEMAP\_NEGATIVEY = &h2000

DDSCAPS2\_CUBEMAP\_POSITIVEZ = &h4000

DDSCAPS2\_CUBEMAP\_NEGATIVEZ = &h8000

DDSCAPS2\_VOLUME = &h200000

End Enum

Class Map

**Map.addTag:**

```

Function addTag(in_meta as meta) As boolean
    if not in_meta.expanded then Return false

    dim allow_sbsp as Boolean = false
    if in_meta.class1 = "psbs" AND (not allow_sbsp) then
        errorbox("Error: Cannot import SBSP tags")
        return false
    end

    //still needs a lot more
    if not cTagTypes.HasKey(reverse(in_meta.class1)) then
        cTagTypes.Value(in_meta.class1) = "Unknown"
    end

    dim temp_meta as new meta
    temp_meta = in_meta.copy

    dim temp_id as integer = new_ID
    if temp_id >= &hFFFFFF then Return false//unable to find any worthwhile IDs
    if temp_id < 3782475776 then Return false//just in case it rolls over

    temp_meta.tagID = temp_id

    //add +'s to the tag name until it's a unique name
    while tagClassNameTable.HasKey(temp_meta.class1 + ":" + temp_meta.name)
        temp_meta.name = temp_meta.name + "+"
    wend

    tags.Append temp_meta
    tagIDTable.Value(temp_meta.tagID) = UBound(tags)
    tagClassNameTable.Value(temp_meta.class1 + ":" + temp_meta.name) = UBound(tags)
    generate_class_folders
    generate_name_folders

    return true
End Function

```

### Map.addTags:

```

Function addTags(in_meta() as meta, sbsp_index as integer, dont_import_duplicates as boolean = false) As boolean
    //this method is specifically for use with the SBSP replacement
    dim problem_tags(-1) as integer
    for i as integer = 0 to in_meta.ubound
        if in_meta(i).map_version <> header.version then
            if in_meta(i).class1 = "mtib" or in_meta(i).class1 = "!dns" then
                if in_meta(i).raw_data.ubound < 0 then
                    problem_tags.append i
                end
            end
        end
    next
    if problem_tags.ubound > -1 then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
    end

```

```

d.ActionButton.Caption="Continue"
d.CancelButton.Visible= True    //show the Cancel button
d.AlternateActionButton.Visible= True    //show the "Internalize" button
d.AlternateActionButton.Caption="Internalize"
d.Message="Warning: References not in target version"
d.Explanation="Certain tags contain references to bitmap and/or sound data from another" + _
" version of Halo. Would you like to continue with the import, internalize the data or cancel?"
b=d.ShowModal    //display the dialog
Select Case b //determine which button was pressed.
Case d.ActionButton
    //user pressed Continue
Case d.AlternateActionButton
    //user pressed Internalize
    dim bitm as Boolean = false
    dim snd as Boolean = false
    for i as integer = 0 to problem_tags.ubound
        if not bitm and in_meta(problem_tags(i)).class1 = "mtib" then
            bitm = true
        end
        if not snd and in_meta(problem_tags(i)).class1 = "!dns" then
            snd = true
        end
        if snd and bitm then exit for i
    next
    if bitm then
        dim internalize_bitmap_f as FolderItem
        Dim dlg as New OpenFileDialog
        dlg.ActionButtonCaption="Select"
        dlg.Title="Bitmaps.map"
        dlg.PromptText="Please select the bitmaps.map to internalize from"
        dlg.Filter=H1_Filetypes.HaloMapFile
        internalize_bitmap_f=dlg.ShowModal()
        if internalize_bitmap_f <> Nil then
            for i as integer = 0 to problem_tags.ubound
                if in_meta(problem_tags(i)).class1 = "mtib" then
                    dim temp_bool as Boolean = H1.internalizeTag(in_meta(problem_tags(i)), internalize_bitmap_f)
                end
            next
        end
    end
    if snd then
        dim internalize_sound_f as FolderItem
        Dim dlg as New OpenFileDialog
        dlg.ActionButtonCaption="Select"
        dlg.Title="Sounds.map"
        dlg.PromptText="Please select the sounds.map to internalize from"
        dlg.Filter=H1_Filetypes.HaloMapFile
        internalize_sound_f=dlg.ShowModal()
        if internalize_sound_f <> Nil then
            for i as integer = 0 to problem_tags.ubound
                if in_meta(problem_tags(i)).class1 = "!dns" then
                    dim temp_bool as Boolean = H1.internalizeTag(in_meta(problem_tags(i)), internalize_sound_f)
                end
            next
        end
    end
end

```

```

        end
    end

    Case d.CancelButton
        //user pressed Cancel
        return false
    End select
end

dim sbsp_metas(-1) as integer
dim sbsp_flag as Boolean = false
dim allow_sbsp as Boolean = false
for i as integer = 0 to UBound(in_meta)
    if not in_meta(i).expanded then Return false
    if in_meta(i).class1 = "psbs" AND (not allow_sbsp) then
        sbsp_metas.append i
        sbsp_flag = true
    end if
next

if sbsp_flag then
    break //serious error
end

dim tag_names(-1) as string
for i as integer = 0 to tags.Ubound
    tag_names.append tags(i).class1 + ":" + tags(i).name
next

dim temp_metas(-1) as meta
for i as integer = 0 to UBound(in_meta)
    dim temp_meta as new meta
    temp_meta = in_meta(i).copy
    temp_metas.append temp_meta
next

dim w as new Progress_Window("Tag Import")
w.tick("No tags imported yet", 0)
w.show

dim skip_metas(-1) as integer

for i as integer = 0 to UBound(temp_metas)

    dim status_num as double = i+1
    dim status_denom as double = UBound(temp_metas)+1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 50.0
    w.tick("Importing Tags", status)

    if sbsp_flag then
        if sbsp_metas.indexOf(i) >= 0 then
            Continue
        end
    end

```



```

end

dim duplicate as boolean = false

dim old_name as string = temp_metas(i).name
dim new_name as string = temp_metas(i).name
while tag_names.indexOf(temp_metas(i).class1 + ":" + new_name) > -1
    new_name = new_name + "+"
    duplicate = true
wend

if duplicate then
    //so this tag already exists within the map
    if dont_import_duplicates then
        //importing the duplicate is not ok
        skip_metas.append i
    else
        //importing the duplicate is ok
        //fix all other references
        for j as integer = 0 to UBound(temp_metas)
            for k as integer = 0 to UBound(temp_metas(j).data.dependencies)
                if temp_metas(j).data.dependencies(k).tag_class = temp_metas(i).class1 AND _
                    temp_metas(j).data.dependencies(k).tag_name = old_name then
                    temp_metas(j).data.dependencies(k).tag_name = new_name
                end
            next
            for k as integer = 0 to UBound(temp_metas(j).data.loneIDs)
                if temp_metas(j).data.loneIDs(k).tag_class = temp_metas(i).class1 AND _
                    temp_metas(j).data.loneIDs(k).tag_name = old_name then
                    temp_metas(j).data.loneIDs(k).tag_name = new_name
                end
            next
        next
        for j as integer = 0 to UBound(tags(sbsp_index).data.loneIDs)
            if tags(sbsp_index).data.loneIDs(j).tag_class = temp_metas(i).class1 AND _
                tags(sbsp_index).data.loneIDs(j).tag_name = old_name then
                tags(sbsp_index).data.loneIDs(j).tag_name = new_name
            end
        next
        for j as integer = 0 to UBound(tags(sbsp_index).data.dependencies)
            if tags(sbsp_index).data.dependencies(j).tag_class = temp_metas(i).class1 AND _
                tags(sbsp_index).data.dependencies(j).tag_name = old_name then
                tags(sbsp_index).data.dependencies(j).tag_name = new_name
            end
        next

        temp_metas(i).name = new_name
        tag_names.append temp_metas(i).class1 + ":" + temp_metas(i).name
    end
end
next

for i as integer = 0 to UBound(temp_metas)

```

```

dim status_num as double = i+1
dim status_denom as double = UBound(temp_metas)+1
dim status_dec as double = status_num/status_denom
dim status as integer = (status_dec * 50.0) + 50
w.tick("Importing Tags", status)

if sbsp_flag then
    if sbsp_metas.IndexOf(i) >= 0 then
        Continue
    end
end

if skip_metas.IndexOf(i) > -1 then Continue //skips any duplicates that are unwanted

if not cTagTypes.HasKey(reverse(temp_metas(i).class1)) then
    cTagTypes.Value(temp_metas(i).class1) = "Unknown"
end

dim temp_id as uint32 = new_ID
if temp_id >= &hFFFFFFF then
    break
w.close
Return false //unable to find any worthwhile IDs
end
if temp_id < 3782475776 then
    break
w.close
Return false//just in case it rolls over
end
temp_metas(i).tagID = temp_id
tags.Append temp_metas(i)
tagIDTable.Value(temp_metas(i).tagID) = UBound(tags) //this should update ID values
tagClassNameTable.Value(temp_metas(i).class1 + ":" + temp_metas(i).name) = UBound(tags)
next
w.close

if sbsp_flag then
    'dim temp_str as string = "Error: the following tags could not be imported:"
    'for i as integer = 0 to UBound(sbsp_metas)
    'temp_str = EndOfLine + temp_metas(sbsp_metas(i)).class1.reverse + ": " + temp_metas(sbsp_metas(i)).name
    'next
    'errorbox(temp_str)
end

generate_class_folders
generate_name_folders

return true
End Function

```

### Map.addTags:

```

Function addTags(in_meta() as meta, dont_import_duplicates as boolean = false) As boolean
    dim problem_tags(-1) as integer

```

```

for i as integer = 0 to in_meta.ubound
    if in_meta(i).map_version <> header.version then
        if in_meta(i).class1 = "mtib" or in_meta(i).class1 = "!dns" then
            if in_meta(i).raw_data.ubound < 0 then
                problem_tags.append i
            end
        end
    end
end
next
if problem_tags.ubound > -1 then
    Dim d as New MessageDialog //declare the MessageDialog object
    Dim b as MessageDialogButton //for handling the result
    d.icon=MessageDialog.GraphicCaution //display warning icon
    d.ActionButton.Caption="Continue"
    d.CancelButton.Visible= True //show the Cancel button
    d.AlternateActionButton.Visible= True //show the "Internalize" button
    d.AlternateActionButton.Caption="Internalize"
    d.Message="Warning: References not in target version"
    d.Explanation="Certain tags contain references to bitmap and/or sound data from another" + _
    " version of Halo. Would you like to continue with the import, internalize the data or cancel?"
    b=d.ShowModal //display the dialog
    Select Case b //determine which button was pressed.
    Case d.ActionButton
        //user pressed Continue
    Case d.AlternateActionButton
        //user pressed Internalize
        dim bitm as Boolean = false
        dim snd as Boolean = false
        for i as integer = 0 to problem_tags.ubound
            if not bitm and in_meta(problem_tags(i)).class1 = "mtib" then
                bitm = true
            end
            if not snd and in_meta(problem_tags(i)).class1 = "!dns" then
                snd = true
            end
            if snd and bitm then exit for i
        next
        if bitm then
            dim internalize_bitmap_f as FolderItem
            Dim dlg as New OpenFileDialog
            dlg.ActionButtonCaption="Select"
            dlg.Title="Bitmaps.map"
            dlg.PromptText="Please select the bitmaps.map to internalize from"
            dlg.Filter=H1_Filetypes.HaloMapFile
            internalize_bitmap_f=dlg.ShowModal()
            if internalize_bitmap_f <> Nil then
                for i as integer = 0 to problem_tags.ubound
                    if in_meta(problem_tags(i)).class1 = "mtib" then
                        dim temp_bool as Boolean = H1.internalizeTag(in_meta(problem_tags(i)), internalize_bitmap_f)
                    end
                next
            end
        end
        if snd then

```

```

    dim internalize_sound_f as FolderItem
    Dim dlg as New OpenFileDialog
    dlg.ActionButtonCaption="Select"
    dlg.Title="Sounds.map"
    dlg.PromptText="Please select the sounds.map to internalize from"
    dlg.Filter=H1_Filetypes.HaloMapFile
    internalize_sound_f=dlg.ShowModal()
    if internalize_sound_f <> Nil then
        for i as integer = 0 to problem_tags.ubound
            if in_meta(problem_tags(i)).class1 = "!dns" then
                dim temp_bool as Boolean = H1.internalizeTag(in_meta(problem_tags(i)), internalize_sound_f)
            end
        next
    end
end

Case d.CancelButton
    //user pressed Cancel
    return false
End select
end

dim sbasp_metas(-1) as integer
dim sbasp_flag as Boolean = false
dim allow_sbasp as Boolean = false
for i as integer = 0 to UBound(in_meta)
    if not in_meta(i).expanded then Return false
    if in_meta(i).class1 = "psbs" AND (not allow_sbasp) then
        sbasp_metas.append i
        sbasp_flag = true
    end if
next

dim tag_names(-1) as string
for i as integer = 0 to tags.Ubound
    tag_names.append tags(i).class1 + ":" + tags(i).name
next

dim temp_metas(-1) as meta
for i as integer = 0 to UBound(in_meta)
    dim temp_meta as new meta
    temp_meta = in_meta(i).copy
    temp_metas.append temp_meta
next

dim w as new Progress_Window("Tag Import")
w.tick("No tags imported yet", 0)
w.show

dim skip_metas(-1) as integer

for i as integer = 0 to UBound(temp_metas)
    if sbasp_flag then
        if sbasp_metas.indexOf(i) >= 0 then

```

```

        Continue
    end
end

dim status_num as double = i+1
dim status_denom as double = UBound(temp_metas)+1
dim status_dec as double = status_num/status_denom
dim status as integer = status_dec * 50.0
w.tick("Importing Tags", status)

dim duplicate as boolean = false

dim old_name as string = temp_metas(i).name
dim new_name as string = temp_metas(i).name
while tag_names.indexOf(temp_metas(i).class1 + ":" + new_name) > -1
    new_name = new_name + "+"
    duplicate = true
wend

if duplicate then
    //so this tag already exists within the map
    if dont_import_duplicates then
        //importing the duplicate is not ok
        skip_metas.append i
    else
        //importing the duplicate is ok
        //fix all other references
        for j as integer = 0 to UBound(temp_metas)
            for k as integer = 0 to UBound(temp_metas(j).data.dependencies)
                if temp_metas(j).data.dependencies(k).tag_class = temp_metas(i).class1 AND _
                    temp_metas(j).data.dependencies(k).tag_name = old_name then
                    temp_metas(j).data.dependencies(k).tag_name = new_name
                end
            next
            for k as integer = 0 to UBound(temp_metas(j).data.loneIDs)
                if temp_metas(j).data.loneIDs(k).tag_class = temp_metas(i).class1 AND _
                    temp_metas(j).data.loneIDs(k).tag_name = old_name then
                    temp_metas(j).data.loneIDs(k).tag_name = new_name
                end
            next
        next
        temp_metas(i).name = new_name
        tag_names.append temp_metas(i).class1 + ":" + temp_metas(i).name
    end
end
next

for i as integer = 0 to UBound(temp_metas)
    if sbasp_flag then
        if sbasp_metas.indexOf(i) >= 0 then
            Continue
        end
    end
    if skip_metas.indexOf(i) > -1 then continue //skips any duplicates that are unwanted

```

```

dim status_num as double = i+1
dim status_denom as double = UBound(temp_metas)+1
dim status_dec as double = status_num/status_denom
dim status as integer = (status_dec * 50.0) + 50
w.tick("Importing Tags", status)

if not cTagTypes.HasKey(reverse(temp_metas(i).class1)) then
    cTagTypes.Value(temp_metas(i).class1) = "Unknown"
end

dim temp_id as uint32 = new_ID
if temp_id >= &hFFFFFFF then
    break
w.close
Return false //unable to find any worthwhile IDs
end
if temp_id < 3782475776 then
    break
w.close
Return false//just in case it rolls over
end
temp_metas(i).tagID = temp_id
tags.Append temp_metas(i)
tagIDTable.Value(temp_metas(i).tagID) = UBound(tags) //this should update ID values
tagClassNameTable.Value(temp_metas(i).class1 + ":" + temp_metas(i).name) = UBound(tags)
next
w.close

if sbasp_flag then
    dim temp_str as string = "Error: the following tags could not be imported:"
    for i as integer = 0 to UBound(sbsp_metas)
        temp_str = EndOfLine + temp_metas(sbsp_metas(i)).class1.reverse + ": " +
            temp_metas(sbsp_metas(i)).name
    next
    errorbox(temp_str)
end

generate_class_folders
generate_name_folders

return true
End Function

```

### Map.change\_name:

```

Sub change_name(tag_index as integer, new_name as string)
    //probably only for expanded maps
    dim old_name as string = tags(tag_index).name
    dim tag_class as string = tags(tag_index).class1
    tags(tag_index).name = new_name

    //fix the dictionary
    if tagClassNameTable.HasKey(tag_class + ":" + old_name) then
        tagClassNameTable.Remove(tag_class + ":" + old_name)
    end
End Sub

```

```

end
tagClassNameTable.Value(tag_class + ":" + new_name) = tag_index

//redo the folders
generate_class_folders
generate_name_folders

dim w as new Progress_Window("Updating References")
w.tick("Beginning update", 0)
w.show
//now check for any tags that are dependent on it and update them
for i as integer = 0 to tags.Ubound
    dim status_num as double = i+1
    dim status_denom as double = UBound(tags)+1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Updating " + tag_class + ": " + new_name,status)

    if tags(i).expanded then
        for j as integer = 0 to tags(i).data.dependencies.ubound
            if tags(i).data.dependencies(j).tag_class = tag_class and _
                tags(i).data.dependencies(j).tag_name = old_name then
                tags(i).data.dependencies(j).tag_name = new_name
            end
        next
        for j as integer = 0 to tags(i).data.loneIDs.ubound
            if tags(i).data.loneIDs(j).tag_class = tag_class and _
                tags(i).data.loneIDs(j).tag_name = old_name then
                tags(i).data.loneIDs(j).tag_name = new_name
            end
        next
    end
next
w.close
End Sub

```

### Map.Constructor:

```

Sub Constructor()
    Header = new Map_Header
    Index_Header = new Map_Index_Header
    redim tags(-1)
    tagIDTable = new Dictionary
    tagClassNameTable = new Dictionary
    cTagTypes = new Dictionary
End Sub

```

### Map.expandMap:

```

Function expandMap() As boolean
    if me.Header.version = 609 then
        errorbox("Error: Can not expand CE Map")
        Return false
    end
    dim e as new map_expander(self)
    e.Run

```

```
Return true
End Function
```

### Map.expandTag:

```
Sub expandTag(index as integer, allow_bsp_reflexives as boolean = true)
    if tags(index).CE_flag = 1 then return //quit it

    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = tags(index).offset
    dim bw as BinaryStream
    dim temp_data(-1) as UInt32
    dim local_magic as integer = magic

    bw = new BinaryStream(tags(index).data.bin)
    bw.LittleEndian = true
    tags(index).data.bin.littleendian = true
    dim data_start as integer = -1
    dim data_size as integer = -1

    if tags(index).class1 <> "psbs" then
        for i as integer = 1 to tags(index).metasize/4
            temp_data.Append br.ReadUInt32
            bw.WriteUInt32(temp_data(i-1))
        next
        tags(index).data.bin.size = tags(index).metasize
        tags(index).data.size = tags(index).metasize
        data_start = tags(index).offset
        data_size = tags(index).metasize
    else
        dim bsp_offset as integer
        dim bsp_size as integer
        for i as integer = 0 to UBound(bsp_data)
            if bsp_data(i).tagID = tags(index).tagID then
                bsp_offset = bsp_data(i).offset
                bsp_size = bsp_data(i).size
                local_magic = bsp_data(i).magic
                exit for i
            end
        next
        br.Position = bsp_offset
        for i as integer = 1 to bsp_size/4
            temp_data.Append br.ReadUInt32
            bw.WriteUInt32(temp_data(i-1))
        next
        tags(index).data.bin.size = bsp_size
        tags(index).data.size = bsp_size
        data_start = 0 //always zero for BSPs since the magic is unmodified by location
        data_size = bsp_size
    end

    //create the target xml folderitem
    'dim xdoc as new XmlDocument
    'xdoc.AppendChild(xdoc.CreateComment(" Eschaton: Meta Structure File "))
```



```

'dim xml_results as XmlNode
'xml_results = xdoc.AppendChild(xdoc.CreateElement("Results"))
'dim xml_map as XmlNode
'xml_map = xml_results.AppendChild(xdoc.CreateElement("Map"))
'xml_map.AppendChild(xdoc.CreateTextNode(header.name))
'dim xml_tag as XmlNode
'xml_tag = xml_results.AppendChild(xdoc.CreateElement("Tag"))
'dim this_class_string as string = tags(index).class1 + tags(index).class2 + tags(index).class3
'this_class_string = this_class_string.ReplaceAll(chr(&hFF), "F")
'xml_tag.AppendChild(xdoc.CreateTextNode(this_class_string))
'dim xml_filename as XmlNode
'xml_filename = xml_results.AppendChild(xdoc.CreateElement("Filename"))
'xml_filename.AppendChild(xdoc.CreateTextNode(tags(index).name))

for i as integer = 0 to UBound(temp_data)
    dim tempint as int32 = temp_data(i)
    if (not(tempint = 0)) AND (not(tempint = &hFFFFFF)) AND (not(tempint = &hCACACACA)) then
        dim tempintmagic as integer = tempint - local_magic
        dim reflexive_count as integer
        dim intTranslation as integer
        if isReflexiveOffset(tempintmagic, data_start, data_size, i*4 + data_start) AND (data_size > 0) then
            if (i >= 1) AND ((tags(index).class1 <> "psbs" AND data_start > 0) OR allow_bsp_reflexives) then
                //reflexive

                //check that the count is non-negative
                //assuming signed integers
                if temp_data(i-1) > 0 then

                    //check that there's a zero at the end
                    dim zero_test as integer = -1
                    if i+1 <= UBound(temp_data) then
                        zero_test = temp_data(i+1)
                    end
                    if zero_test = 0 then //reflexives need to have a chunk count and a set of zeros following them
                        intTranslation = tempintmagic - data_start

                        'dim ref as xmlnode
                        'dim location as xmlnode
                        'dim translation as xmlnode
                        'dim chunkcount as xmlnode
                        'ref = xml_results.AppendChild(xdoc.CreateElement("Reflexive"))
                        'location = ref.AppendChild(xdoc.CreateElement("Location"))
                        'location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
                        'translation = ref.AppendChild(xdoc.CreateElement("Translation"))
                        'translation.AppendChild(xdoc.CreateTextNode("0x" + hex(intTranslation) ))
                        'chunkcount = ref.AppendChild(xdoc.CreateElement("ChunkCount"))
                        'chunkcount.AppendChild(xdoc.CreateTextNode(str(temp_data(i-1))))

                        //simpler method not involving xml
                        dim temp_reflexive as new h1.reflexive
                        temp_reflexive.offset = (i-1)*4
                        temp_reflexive.count = temp_data(i-1)
                        temp_reflexive.translation = intTranslation
                        tags(index).data.reflexives.append temp_reflexive
                    end if
                end if
            end if
        end if
    end if
end for

```

```

        end
    end
end
else

if(tagIDTable.HasKey(temp_data(i))) then
    //either a loneID or a dependency
    dim item as new meta
    item = tags(tagIDTable.Value(temp_data(i)))

    'dim dep as xmlnode
    'dim location as xmlnode
    'dim tagclass as xmlnode
    'dim filename as xmlnode
    if (i >= 3) AND isTag(temp_data(i-3)) then
        '//dependency
        'dep = xml_results.AppendChild(xdoc.CreateElement("Dependency"))
        'location = dep.AppendChild(xdoc.CreateElement("Location"))
        'location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
        'tagclass = dep.AppendChild(xdoc.CreateElement("Tagclass"))
        'dim class_string as string = item.class1 + item.class2 + item.class3
        'class_string = class_string.ReplaceAll(chr(&hFF), "F")
        'tagclass.AppendChild(xdoc.CreateTextNode(class_string))
        'filename = dep.AppendChild(xdoc.CreateElement("Filename"))
        'filename.AppendChild(xdoc.CreateTextNode(item.name))

        dim temp_dependency as new H1.tag_reference
        temp_dependency.offset = (i-3)*4
        temp_dependency.tag_class = item.class1
        temp_dependency.tag_name = item.name
        tags(index).data.dependencies.append temp_dependency

    else
        if tags(index).class1 <> "psbs" OR true then //was originally checking if sbps loneID updating
            was breaking
            'dep = xml_results.AppendChild(xdoc.CreateElement("LoneID"))
            'location = dep.AppendChild(xdoc.CreateElement("Location"))
            'location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
            'tagclass = dep.AppendChild(xdoc.CreateElement("Tagclass"))
            'dim class_string as string = item.class1 + item.class2 + item.class3
            'class_string = class_string.ReplaceAll(chr(&hFF), "F")
            'tagclass.AppendChild(xdoc.CreateTextNode(class_string))
            'filename = dep.AppendChild(xdoc.CreateElement("Filename"))
            'filename.AppendChild(xdoc.CreateTextNode(item.name))

            dim temp_loneID as new H1.tag_reference
            temp_loneID.offset = i*4
            temp_loneID.tag_class = item.class1
            temp_loneID.tag_name = item.name
            tags(index).data.loneIDs.append temp_loneID
        end
    end
end
end

```

```

end

else
  if tempint = &hFFFFFF AND (i >= 3) AND isTag(temp_data(i-3)) then
    dim hex_str as string = hex(temp_data(i-3))
    while hex_str.len < 8
      hex_str = "0" + hex_str
    wend

    dim class_str as string = chr(val("&h" + hex_str.mid(1,2))) + chr(val("&h" + hex_str.mid(3,2))) _
    + chr(val("&h" + hex_str.mid(5,2))) + chr(val("&h" + hex_str.mid(7,2)))
    dim temp_dependency as new H1.tag_reference
    temp_dependency.offset = (i-3)*4
    temp_dependency.tag_class = class_str.reverse
    temp_dependency.tag_name = "nulled out"
    tags(index).data.dependencies.append temp_dependency
  end
end
next

select case tags(index).class1

case "psbs"
  //break

case "2dom"
  //extract all the raw data for vert and inds
  dim m as new mod2_class(fs, tags(index).offset, magic)
  m.read
  for i as integer = 0 to UBound(m.geometries)
    for j as integer = 0 to UBound(m.geometries(i).parts)
      dim temp_raw_ind as new Meta_Raw
      dim raw_writer as binarystream
      raw_writer = new BinaryStream( temp_raw_ind.bin )
      temp_raw_ind.bin.size = (m.geometries(i).parts(j).ind_count+2) * 2
      temp_raw_ind.size = (m.geometries(i).parts(j).ind_count+2) * 2
      br.Position = m.geometries(i).parts(j).ind_offset1 + Index_Header.ind_offset + Index_Header.vert_offset
      dim ind_pos as integer = br.Position
      for k as integer = 1 to (m.geometries(i).parts(j).ind_count+2) * 2
        raw_writer.WriteByte(br.ReadByte)
      next
      temp_raw_ind.str_info = "mod2:geometry:" + str(i) + ":part:" + str(j) + ":ind"
      temp_raw_ind.offset = ind_pos
      tags(index).raw_data.append temp_raw_ind
      raw_writer.close
      dim temp_raw_vert as new Meta_Raw
      raw_writer = new BinaryStream( temp_raw_vert.bin )
      temp_raw_vert.bin.size = m.geometries(i).parts(j).vert_count * &h44
      temp_raw_vert.size = m.geometries(i).parts(j).vert_count * &h44
      br.Position = m.geometries(i).parts(j).vert_offset + Index_Header.vert_offset
      dim vert_pos as integer = br.Position
      for k as integer = 1 to m.geometries(i).parts(j).vert_count * &h44
        raw_writer.WriteByte(br.ReadByte)
      next
    next
  next

```

```

        temp_raw_vert.str_info = "mod2:geometry:" + str(i) + ":part:" + str(j) + ":vert"
        temp_raw_vert.offset = vert_pos
        tags(index).raw_data.append temp_raw_vert
    next
next
case "!dns"
    //extract the raw data for sounds if internalized
    dim s as new snd_class_RW(fs, fs, tags(index).offset, magic)
    s.read
    for i as integer = 0 to UBound(s.pitch)
        for j as integer = 0 to UBound(s.pitch(i).permutations)
            if s.pitch(i).permutations(j).internal then
                dim temp_raw as new Meta_Raw
                dim raw_writer as new BinaryStream( temp_raw.bin )
                temp_raw.bin.size = s.pitch(i).permutations(j).size
                temp_raw.size = s.pitch(i).permutations(j).size
                br.Position = s.pitch(i).permutations(j).offset
                dim temp_pos as integer = br.Position
                for k as integer = 1 to s.pitch(i).permutations(j).size
                    raw_writer.WriteByte(br.ReadByte)
                next
                temp_raw.str_info = "snd!:pitch:" + str(i) + ":permutation:" + str(j)
                temp_raw.offset = temp_pos
                tags(index).raw_data.append temp_raw
            end
        next
    next
case "mtib"
    //extract the raw data for bitmaps if internalized
    dim b as new bitmap_class_RW(fs, fs, tags(index).offset, magic)
    b.read
    for i as integer = 0 to UBound(b.image_info)
        if not (bitand(b.image_info(i).flags, &h100) = &h100) then
            //internal bitmap data
            dim temp_raw as new Meta_Raw
            dim raw_writer as new BinaryStream( temp_raw.bin )
            temp_raw.bin.size = b.image_info(i).size
            temp_raw.size = b.image_info(i).size
            br.position = b.image_info(i).offset
            dim temp_pos as integer = br.Position
            for j as integer = 1 to b.image_info(i).size
                raw_writer.writebyte(br.readbyte)
            next
            temp_raw.str_info = "bitm:image_info:" + str(i)
            temp_raw.offset = temp_pos
            tags(index).raw_data.append temp_raw
        end
    next
end select

br.Close
bw.Close

```

```

dim data_writer as new BinaryStream(tags(index).data.bin)
data_writer.LittleEndian = true
data_writer.position = 0
tags(index).data.bin.size = 4*(UBound(temp_data)+1)
for i as integer = 0 to UBound(temp_data)
    data_writer.Writeuint32(temp_data(i))
next
'tags(index).data.xdoc = xdoc
tags(index).expanded = true
tags(index).magic = local_magic
End Sub

```

### **Map.generate\_class\_folders:**

```

Sub generate_class_folders()
    dim temp_dic as new Dictionary
    meta_class_folders = new vFolder
    dim temp_folders(-1) as vFolder

    for i as integer = 0 to UBound(tags)
        if temp_dic.HasKey(tags(i).class1) then
            temp_folders(temp_dic.Value(tags(i).class1)).item.Append i
        else
            dim temp_folder as new vFolder
            dim temp_str as string = "Unknown"
            if cTagTypes.HasKey(reverse(tags(i).class1)) then
                temp_str = cTagTypes.Value(reverse(tags(i).class1))
            end
            temp_folder.name = reverse(tags(i).class1) + ": " + temp_str
            temp_folder.item.Append i
            temp_folders.Append temp_folder
            temp_dic.Value(tags(i).class1) = UBound(temp_folders)
        end
    next

    meta_class_folders.child = temp_folders
End Sub

```

### **Map.generate\_name\_folders:**

```

Sub generate_name_folders()
    meta_name_folders = new vFolder

    for i as integer = 0 to UBound(tags)
        //loop through all of the tags
        dim path() as string = split(tags(i).name, "\")
        dim current_folder as new vFolder
        current_folder = meta_name_folders

        for j as integer = 0 to UBound(path) - 1
            dim found_folder as boolean = false
            for k as integer = 0 to UBound(current_folder.child)
                if current_folder.child(k).name = path(j) then
                    found_folder = true
                    current_folder = current_folder.child(k)
                exit for k
            next k
        next j
    next i

```

```

        end
    next
    if not found_folder then
        dim temp_folder as new vFolder
        temp_folder.name = path(j)
        current_folder.child.Append temp_folder
        current_folder = current_folder.child(UBound(current_folder.child))
    end
next

    current_folder.item.Append i
next
End Sub

```

## Map.initCTagTypes:

```

Sub initCTagTypes()
    cTagTypes = new Dictionary

    cTagTypes.value("actr")="Actor"
    cTagTypes.value("actv")="Actor Variant"
    cTagTypes.value("ant!")="Antenna"
    cTagTypes.value("antr")="Model Animations"
    cTagTypes.value("bipd")="Biped"
    cTagTypes.value("bitm")="Bitmap"
    cTagTypes.value("boom")="Spheroid"
    cTagTypes.value("cdmg")="Continuous Damage Effect"
    cTagTypes.value("coll")="Model Collision Geometry"
    cTagTypes.value("colo")="Color Table"
    cTagTypes.value("cont")="Contrail"
    cTagTypes.value("ctrl")="Device Control"
    cTagTypes.value("deca")="Decal"
    cTagTypes.value("DeLa")="UI Widget Definition"
    cTagTypes.value("devc")="Input Device Defaults"
    cTagTypes.value("devi")="Device"
    cTagTypes.value("dobc")="Detail Object Collection"
    cTagTypes.value("effe")="Effect"
    cTagTypes.value("elec")="Lightning"
    cTagTypes.value("equip")="Equipment"
    cTagTypes.value("flag")="Flag"
    cTagTypes.value("fog ")="Fog"
    cTagTypes.value("font")="Font"
    cTagTypes.value("foot")="Material Effects"
    cTagTypes.value("garb")="Garbage"
    cTagTypes.value("glw!")="Glow"
    cTagTypes.value("grhi")="Grenade HUD Interface"
    cTagTypes.value("hmt ")="HUD Message Text"
    cTagTypes.value("hud#")="HUD Number"
    cTagTypes.value("hudg")="HUD Globals"
    cTagTypes.value("item")="Item"
    cTagTypes.value("itmc")="Item collection"
    cTagTypes.value("jpt!")="Damage Effect"
    cTagTypes.value("lens")="Lens Flare"
    cTagTypes.value("lifi")="Device Light Fixture"
    cTagTypes.value("ligh")="Light"

```

```

cTagTypes.value("lsnd")="Sound Looping"
cTagTypes.value("mach")="Device Machine"
cTagTypes.value("matg")="Globals"
cTagTypes.value("metr")="Meter"
cTagTypes.value("mgs2")="Light Volume"
cTagTypes.value("mod2")="Model (PC)"
cTagTypes.value("mode")="Model (Xbox)"
cTagTypes.value("mply")="Multiplayer Scenario Description"
cTagTypes.value("ngpr")="Preferences Network Game"
cTagTypes.value("obje")="Object"
cTagTypes.value("part")="Particle"
cTagTypes.value("pctl")="Particle System"
cTagTypes.value("phys")="Physics"
cTagTypes.value("plac")="Placeholder"
cTagTypes.value("pphy")="Point Physics"
cTagTypes.value("proj")="Projectile"
cTagTypes.value("rain")="Weather Particle System"
cTagTypes.value("sbsp")="Scenario Structure BSP"
cTagTypes.value("scen")="Scenery"
cTagTypes.value("scex")="Shader Transparent Chicago Extended"
cTagTypes.value("schi")="Shader Transparent Chicago"
cTagTypes.value("scnr")="Scenario"
cTagTypes.value("senv")="Shader Environment"
cTagTypes.value("sgla")="Shader Transparent Glass"
cTagTypes.value("shdr")="Shader"
cTagTypes.value("sky ")="Sky"
cTagTypes.value("smet")="Shader Transparent Meter"
cTagTypes.value("snd!")="Sound"
cTagTypes.value("snde")="Sound Environment"
cTagTypes.value("soso")="Shader Model"
cTagTypes.value("sotr")="Shader Transparent Generic"
cTagTypes.value("Soul")="UI Widget Collection"
cTagTypes.value("spla")="Shader Transparent Plasma"
cTagTypes.value("ssce")="Sound Scenery"
cTagTypes.value("str#")="String List"
cTagTypes.value("swat")="Shader Transparent Water"
cTagTypes.value("tagc")="Tag Collection"
cTagTypes.value("trak")="Camera Track"
cTagTypes.value("udlg")="Dialogue"
cTagTypes.value("unhi")="Unit HUD Interface"
cTagTypes.value("unit")="Unit"
cTagTypes.value("ustr")="Unicode String List"
cTagTypes.value("vehi")="Vehicle"
cTagTypes.value("vcky")="Virtual Keyboard"
cTagTypes.value("weap")="Weapon"
cTagTypes.value("wind")="Wind"
cTagTypes.value("wphi")="Weapon HUD Interface"

```

End Sub

### Map.internalizeTag:

```

Function internalizeTag(index as integer, data_f as folderItem) As boolean
    if tags(index).CE_flag = 1 then return false//quit it
    if not tags(index).expanded then return false

```

```

//theoretically this should work
dim return_bool as Boolean = true

select case tags(index).class1
case "!dns"
    //extract the raw data for sounds if external
    dim s as new snd_class_EX(tags(index).data.bin, tags(index).offset, magic)
    s.read
    for i as integer = 0 to UBound(s.pitch)
        for j as integer = 0 to UBound(s.pitch(i).permutations)
            if not s.pitch(i).permutations(j).internal then
                dim temp_raw as new Meta_Raw
                dim raw_writer as new BinaryStream( temp_raw.bin )
                temp_raw.bin.size = s.pitch(i).permutations(j).size
                temp_raw.size = s.pitch(i).permutations(j).size
                dim br as BinaryStream = data_f.OpenAsBinaryFile
                br.LittleEndian = true
                br.Position = s.pitch(i).permutations(j).offset
                for k as integer = 1 to s.pitch(i).permutations(j).size
                    raw_writer.WriteByte(br.ReadByte)
                next
                temp_raw.str_info = "snd!:pitch:" + str(i) + ":permutation:" + str(j)
                dim temp_bool as boolean = s.update_offset(i, j, s.pitch(i).permutations(j).offset, true)
                if temp_bool then
                    tags(index).raw_data.append temp_raw
                else
                    return_bool = false
                end
                br.Close
            end
        next
    next
case "mtib"
    //extract the raw data for bitmaps if internalized
    dim b as new bitmap_class_EX(tags(index).data.bin, tags(index).offset, magic)
    b.read
    for i as integer = 0 to UBound(b.image_info)
        if (bitand(b.image_info(i).flags, &h100) = &h100) then
            //external bitmap data
            dim temp_raw as new Meta_Raw
            dim raw_writer as new BinaryStream( temp_raw.bin )
            temp_raw.bin.size = b.image_info(i).size
            temp_raw.size = b.image_info(i).size
            dim br as BinaryStream = data_f.OpenAsBinaryFile
            br.LittleEndian = true
            br.position = b.image_info(i).offset
            for j as integer = 1 to b.image_info(i).size
                raw_writer.writebyte(br.readbyte)
            next
            temp_raw.str_info = "bitm:image_info:" + str(i)
            dim temp_bool as boolean = b.update_offset(i, b.image_info(i).offset, true)
            if temp_bool then
                tags(index).raw_data.append temp_raw
            else

```



```

        return_bool = false
    end
    br.Close
end
next
end select

return return_bool
End Function

```

### Map.isOffset:

```

Function isOffset(intOffset as integer, tagoffset as integer, tagMetasize as integer) As boolean
    //not correct, also needs to check to see if it's in the string table
    return ((intOffset >= header.TagIndexOffset) AND (intOffset <= header.decomp_len))
End Function

```

### Map.isReflexiveOffset:

```

Function isReflexiveOffset(intOffset as integer, tagoffset as integer, tagMetasize as integer, currentOffset as integer) As boolean
    dim bool1 as Boolean = (intOffset >= tagoffset)
    dim bool2 as Boolean = (intOffset <= (tagoffset + tagMetasize))
    dim bool3 as Boolean = (intOffset > currentOffset)
    return (intOffset >= tagoffset) AND (intOffset <= (tagoffset + tagMetasize)) AND (intOffset > currentOffset)
End Function

```

### Map.isTag:

```

Function isTag(i as int32) As boolean
    dim hex_str as string = hex(i)
    while hex_str.len < 8
        hex_str = "0" + hex_str
    wend

    dim class_str as string = chr(val("&h" + hex_str.mid(1,2))) + chr(val("&h" + hex_str.mid(3,2))) _
    + chr(val("&h" + hex_str.mid(5,2))) + chr(val("&h" + hex_str.mid(7,2)))

    return cTagTypes.HasKey(class_str)
End Function

```

### Map.new\_ID:

```

Function new_ID() As uint32
    dim temp_id as uint32 = &hFFFFFFF
    for start as uint32 = 3782475776 to &hFFFFFFFE
        for i as uint32 = start to &hFFFFFFF step 65537
            if not tagIDTable.HasKey(i) AND i <> &hFFFFFFF AND i >= 3782475776 then
                temp_id = i
                exit for start
            end
        next
    next

    return temp_id
End Function

```

Map.read:

```
Function read(f as folderItem) As boolean
    fs = f
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = 0
    if not header.read(br) then
        return false
    end
    br.Position = header.TagIndexOffset
    Index_Header.read(br, header.version <> 5)
    magic = Index_Header.index_magic - (header.TagIndexOffset + 36)
    if (Header.version <> 5) then
        magic = magic - 4
    end

    redim tags(-1)
    tagIDTable = new Dictionary
    tagClassNameTable = new Dictionary

    for i as integer = 1 to index_header.tagcount
        dim temp_meta as new meta
        temp_meta.map_version = header.version
        temp_meta.map_name = header.name
        temp_meta.read(br, magic)
        tags.Append(temp_meta)
        tagIDTable.Value(temp_meta.tagID) = UBound(tags)
        tagClassNameTable.Value(_
            temp_meta.class1 + ":" + temp_meta.name) _
            = UBound(tags)
    next

    //get the relevant bsp data
    redim bsp_data(-1)
    if tagIDTable.HasKey(index_header.BaseTag) then
        dim temp_index_item as new meta
        temp_index_item = tags(tagIDTable.Value(index_header.BaseTag))
        base_scnr = tagIDTable.Value(index_header.BaseTag)
        br.Position = temp_index_item.offset + &h5A4 //offset to the scenario reflexive
        dim count as integer = br.ReadInt32
        dim offset as integer = br.ReadInt32 - magic

        br.Position = offset
        for i as integer = 1 to Count
            dim temp_bsp as new bsp_scenario_data
            temp_bsp.read(br, magic)
            bsp_data.Append(temp_bsp)
            if tagIDTable.HasKey(temp_bsp.tagID) then
                tags(tagIDTable.Value(temp_bsp.tagID)).offset = temp_bsp.offset
            end
        next
    end

    //small hack to fix order
```

```

dim orig(-1) as integer
orig = tags.SortbyOffset()

for i as integer = 1 to UBound(tags)
    tags(i-1).metasize = tags(i).offset - tags(i-1).offset
    if i > 1 then
        if tags(i-1).offset < 0 then
            tags(i-2).metasize = tags(i).offset - tags(i-2).offset
        end
    end
    if tags(i-1).CE_flag = 1 then tags(i-1).metasize = 0
    if tags(i-1).class1 = "psbs" then
        for j as integer = 0 to UBound(bsp_data)
            if bsp_data(j).tagID = tags(i-1).tagID then
                tags(i-1).metasize = bsp_data(j).size
            end
        next
    end
end
next

//now do a quick once over for tags that might have been meta swapped
for i as integer = 0 to UBound(tags)
    if tags(i).CE_flag = 0 AND tags(i).class1 <> "psbs" AND tags(i).metasize = 0 then
        for j as integer = i+1 to UBound(tags)
            //this loops through all the tags after
            //if a tag has the same offset and is a valid file with actual meta data size
            if tags(j).offset = tags(i).offset AND tags(j).metasize > 0 _
                AND tags(j).CE_flag = 0 AND tags(j).class1 <> "psbs" then
                tags(i).metasize = tags(j).metasize
            end
            if tags(j).offset > tags(i).offset then
                exit for j
            end
        next
    end
end
next

tags(UBound(tags)).metasize = br.Length - tags(UBound(tags)).offset
if tags(UBound(tags)).CE_flag = 1 then tags(UBound(tags)).metasize = 0
if tags(ubound(tags)).class1 = "psbs" then
    for j as integer = 0 to UBound(bsp_data)
        if bsp_data(j).tagID = tags(ubound(tags)).tagID then
            tags(ubound(tags)).metasize = bsp_data(j).size
        end
    next
end

//now correct the tag order
orig.SortWith(tags)

initCTagTypes
for i as integer = 0 to UBound(tags)
    if not cTagTypes.HasKey(reverse(tags(i).class1)) then
        cTagTypes.Value(reverse(tags(i).class1)) = "Unknown"
    end
end

```

```

    end
next

generate_class_folders
generate_name_folders
return true
End Function

```

### Map.recursive\_tags:

```

Function recursive_tags(tag_index as integer, index_list() as integer) As integer()
//only for use on tags already in a map directory
if not tags(tag_index).expanded then
//nothing to read for indexed tags
if tags(tag_index).CE_flag = 1 then return index_list

    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = tags(tag_index).offset

    dim temp_list(-1) as integer
    while br.Position < tags(tag_index).offset + tags(tag_index).metasize
        dim temp_position as integer = br.position
        dim temp_val as UInt32 = br.ReadUInt32
        if tagIDTable.haskey(temp_val) then
            temp_list.append tagIDTable.Value(temp_val)
        end
    wend
    br.Close

    temp_list = remove_redundant_ints(temp_list)
    temp_list = remove_redundant_ints_other(index_list, temp_list)

    for i as integer = 0 to UBound(temp_list)
        index_list.Append temp_list(i)
    next

    for i as integer = 0 to UBound(temp_list)
        index_list = recursive_tags(temp_list(i), index_list)
    next

    index_list = remove_redundant_ints(index_list)

    return index_list
else
    dim temp_list(-1) as integer
    for i as integer = 0 to UBound(tags(tag_index).data.dependencies)
        dim tagclass as string = tags(tag_index).data.dependencies(i).tag_class
        dim filename as string = tags(tag_index).data.dependencies(i).tag_name
        if tagclass <> "psbs" and tagClassNameTable.HasKey(tagclass + ":" + filename) then
            dim temp_int as integer = tagClassNameTable.value(tagclass + ":" + filename)
            temp_list.append temp_int
        end
    next
    for i as integer = 0 to UBound(tags(tag_index).data.loneIDs)

```

```

        dim tagclass as string = tags(tag_index).data.loneIDs(i).tag_class
        dim filename as string = tags(tag_index).data.loneIDs(i).tag_name
        if tagclass <> "psbs" and tagClassNameTable.HasKey(tagclass + ":" + filename) then
            dim temp_int as integer = tagClassNameTable.value(tagclass + ":" + filename)
            temp_list.append temp_int
        end
    next
    temp_list = remove_redundant_ints(temp_list)
    temp_list = remove_redundant_ints_other(index_list, temp_list)

    for i as integer = 0 to UBound(temp_list)
        index_list.Append temp_list(i)
    next

    for i as integer = 0 to UBound(temp_list)
        index_list = recursive_tags(temp_list(i), index_list)
    next

    index_list = remove_redundant_ints(index_list)

    return index_list
end
End Function

```

### Map.remove\_tag:

```

Function remove_tag(tag_index as integer) As boolean
    //now for the dangerous process of removing a tag
    if not expanded then return false
    if tag_index < 0 or tag_index > tags.Ubound then return false
    dim class_str as string = tags(tag_index).class1
    dim name_str as string = tags(tag_index).name

    //check to make sure it's not valuable and can't be replaced
    //some tags still will slip through but this covers the big ones
    if tag_index = base_scnr then
        errorbox("Error: Cannot delete this SCNR tag")
        return false
    end
    for i as integer = 0 to bsp_data.Ubound
        if tagIDTable.HasKey(bsp_data(i).tagID) then
            if tagIDTable.Value(bsp_data(i).tagID) = tag_index then
                errorbox("Error: Cannot delete this SBSP tag")
                return false
            end
        end
    next

    //remove all references to that tag
    for i as integer = 0 to tags.Ubound
        for j as integer = 0 to tags(i).data.dependencies.Ubound
            if tags(i).data.dependencies(j).tag_class = class_str AND _
                tags(i).data.dependencies(j).tag_name = name_str then
                tags(i).data.dependencies(j).tag_name = "nulled out"
            end
        next
    next
end

```

```

    next
    for j as integer = 0 to tags(i).data.loneIDs.Ubound
        if tags(i).data.loneIDs(j).tag_class = class_str AND _
            tags(i).data.loneIDs(j).tag_name = name_str then
            tags(i).data.loneIDs(j).tag_name = "nulled out"
        end
    next
next

//kill it
tags.Remove(tag_index)

tagIDTable = new Dictionary
tagClassNameTable = new Dictionary

for i as integer = 0 to UBound(tags)
    tagIDTable.Value(tags(i).tagID) = i
    tagClassNameTable.Value(_
        tags(i).class1 + ":" + tags(i).name) _
        = i
next

generate_class_folders
generate_name_folders

return true
End Function

```

### Map.reorder\_tags:

Sub reorder\_tags(remove\_unreferenced as boolean = false)  
 //this method is designed to reorder the structure of the Halo map file to maintain the standard order

```

'scnr
'matg
'tagc (all scenario)
'bitm (loading background what?)
'ustr (loading string?)
'ustr (mp map list?)
'bitm (lag icon)
'snd! (ui cursor sound)
'snd! (ui forward sound)
'snd! (ui backward sound)
'tagc (mp scenario)
'sbsp

//first need to do all the BSP related tags in order so that they can be removed from scenario listing
//BLEEEEEEEEEAAAAAAARRRRRRGH!
'dim sbsp_list(-1) as integer
'for i as integer = 0 to UBound(bsp_data)
'dim temp_list(-1) as integer
'if tagIDTable.HasKey(bsp_data(i).tagID) then
'dim sbsp_index as integer = tagIDTable.value(bsp_data(i).tagID)
'temp_list.append sbsp_index
'temp_list = recursive_tags(sbsp_index, temp_list)

```

```

'for j as integer = 0 to UBound(temp_list)
'sbsp_list.Append temp_list(j)
'next
'sbsp_list = remove_redundant_ints(sbsp_list)
'end
'next

dim w as new Progress_Window("Reorganizing Tags")
w.tick("Resolving references", 0)

//now do the tags in order
dim master_list(-1) as integer

//scnr tag
if base_scnr > -1 and base_scnr <= tags.ubound then
    dim temp_list(-1) as integer
    temp_list.append base_scnr
    temp_list = recursive_tags(base_scnr, temp_list)
    for i as integer = 0 to UBound(temp_list)
        master_list.Append temp_list(i)
    next
    master_list = remove_redundant_ints(master_list)
    //master_list = remove_redundant_ints_other(sbsp_list, master_list)
else
    break
end

dim tag_string() as string
//matg
tag_string.append "gtam:globals\globals"

//tagc
tag_string.append "cgat:ui\ui_tags_loaded_all_scenario_types"

//bitm background
tag_string.append "mtib:ui\shell\bitmaps\background"

//ustr loading
tag_string.append "rtsu:ui\shell\strings\loading"

//ustr mp map list
tag_string.append "rtsu:ui\shell\main_menu\mp_map_list"

//bitm lag
tag_string.append "mtib:ui\shell\bitmaps\trouble_brewing"

//snd ui cursor
tag_string.append "!dns:sound\sfx\ui\cursor"

//snd ui forward
tag_string.append "!dns:sound\sfx\ui\forward"

//snd ui back
tag_string.append "!dns:sound\sfx\ui\back"

```

```

//tagc maptype scenario type
select case Header.MapType
case 0 //single player
    tag_string.append "cgat:ui\ui_tags_loaded_solo_scenario_type"
case 1 //multiplayer
    tag_string.append "cgat:ui\ui_tags_loaded_multiplayer_scenario_type"
case 2 //UI
    tag_string.append "cgat:ui\ui_tags_loaded_mainmenu_scenario_type"
case else
    break
end select

for i as integer = 0 to UBound(tag_string)
    dim status_num as double = master_list.ubound + 1
    dim status_denom as double = tags.ubound + 1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Resolving references", status)

    if tagClassNameTable.HasKey(tag_string(i)) then
        dim temp_list(-1) as integer
        temp_list.append tagClassNameTable.value(tag_string(i))
        temp_list = recursive_tags(tagClassNameTable.Value(tag_string(i)), temp_list)
        for j as integer = 0 to UBound(temp_list)
            master_list.Append temp_list(j)
        next
        master_list = remove_redundant_ints(master_list)
    else
        break
    end
next

dim sbsp_list(-1) as integer
for i as integer = 0 to UBound(bsp_data)
    dim temp_list(-1) as integer
    if tagIDTable.HasKey(bsp_data(i).tagID) then
        dim sbsp_index as integer = tagIDTable.value(bsp_data(i).tagID)
        temp_list.append sbsp_index
        temp_list = recursive_tags(sbsp_index, temp_list)
        for j as integer = 0 to UBound(temp_list)
            sbsp_list.Append temp_list(j)
        next
        sbsp_list = remove_redundant_ints(sbsp_list)
    end
next

for i as integer = 0 to UBound(sbsp_list)
    dim status_num as double = master_list.ubound + 1
    dim status_denom as double = tags.ubound + 1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Resolving references", status)

```



```

    master_list.Append sbasp_list(i)
next
master_list = remove_redundant_ints(master_list)

if not remove_unreferenced then
    //tack on all the tags, remove redundant will get rid of everything that is already there
    //only if they are needed
    dim status_num as double = master_list.ubound + 1
    dim status_denom as double = tags.ubound + 1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Resolving references", status)

    dim temp_list(-1) as integer
    for i as integer = 0 to UBound(tags)
        temp_list.Append i
    next
    temp_list = remove_redundant_ints_other(master_list, temp_list)
    for i as integer = 0 to UBound(temp_list)
        master_list.append temp_list(i)
    next
    master_list = remove_redundant_ints(master_list)
end

//now for the true silliness!
//need some method by which all of the tags can be properly ordered
w.tick("Resolving references", 100)
dim tag_count as integer = tags.UBound
dim insert_count as integer = 0
for i as integer = 0 to master_list.ubound
    tags.Insert(i, tags(master_list(i) + insert_count))
    insert_count = insert_count + 1
next
//take care of extra tags on the end
if remove_unreferenced then
    tag_count = master_list.ubound
end
redim tags(tag_count)
w.close

tagIDTable = new Dictionary
tagClassNameTable = new Dictionary

for i as integer = 0 to UBound(tags)
    tagIDTable.Value(tags(i).tagID) = i
    tagClassNameTable.Value(_
        tags(i).class1 + ":" + tags(i).name) _
        = i
next

generate_class_folders
generate_name_folders
End Sub

```

## Map.replaceSBSP:

Function replaceSBSP(replacementIndex as integer, new\_meta as meta\_pack, no\_duplicates as boolean = false) As boolean

```
if not me.expanded then return false //can't replace a tag if the map isn't expanded
if replacementIndex < 0 OR replacementIndex > UBound(tags) then
    //bad index
    return false
end
if tags(replacementIndex).class1 <> "psbs" then return false //big time error
if new_meta.class1 <> "psbs" then return false //can't use a non-sbsp meta pack
if new_meta.metas(0).class1 <> "psbs" then return false //something went majorly wrong
for i as integer = 1 to UBound(new_meta.metas)
    if new_meta.metas(i).class1 = "psbs" then
        //can only replace one bsp at a time
        return false
    end
next
```

```
dim bsp_data_index as integer = -1
for i as integer = 0 to UBound(bsp_data)
    if tags(replacementIndex).tagID = bsp_data(i).tagID then
        bsp_data_index = i
        exit for i
    end
next
if bsp_data_index = -1 then return false //problem here as well
```

```
dim new_bsp as new meta
new_bsp = new_meta.metas(0).copy
```

```
dim convert_magic as boolean = false
```

```
//check to see whether or not to convert BSP magic
//the problem comes from extracting BSP info I think. something is going wrong with either the construction
//of the xml file, or in it's subsequent implementation
//SBSPs really are a pain in the ass
#if DebugBuild
```

```
    if (new_bsp.map_version = 6 AND me.Header.version <> 6) _
        OR (new_bsp.map_version <> 6 AND me.Header.version = 6) then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Convert"
        d.CancelButton.Visible= True //show the Cancel button
        d.AlternateActionButton.Visible= True //show the "Don't Save" button
        d.AlternateActionButton.Caption="Don't Convert"
        d.Message="Warning: This SBSP tag has an incompatible magic value"
        d.Explanation="Would you like to convert the magic value? (This may partially corrupt the tag)"
        b=d.ShowModal //display the dialog
        Select Case b //determine which button was pressed.
        Case d.ActionButton
            convert_magic = true
        Case d.AlternateActionButton
            convert_magic = false
        End Select
    end if
end if
```

```

        Case d.CancelButton
            return false//cancel the process
        End select
    end
#else
    if (new_bsp.map_version = 6 AND me.Header.version <> 6) _
        OR (new_bsp.map_version <> 6 AND me.Header.version = 6) then
        errorbox("Error: this BSP is incompatible with the target map")
        return false
    end
#endif

dim new_magic as integer
dim old_magic as integer
if new_bsp.map_version = 6 then
    old_magic = &h4D610000 - new_bsp.data.size
else
    old_magic = &h41B40000 - new_bsp.data.size
end
if convert_magic then
    new_bsp.map_version = me.Header.version
end
if new_bsp.map_version = 6 then
    new_magic = &h4D610000 - new_bsp.data.size
else
    new_magic = &h41B40000 - new_bsp.data.size
end
dim new_size as integer = new_bsp.data.size

//ok, now we do the deed
bsp_data(bsp_data_index).magic = new_magic
bsp_data(bsp_data_index).size = new_size
new_bsp.magic = new_magic
new_bsp.tagID = tags(replacementIndex).tagID
tags(replacementIndex) = new_bsp
tagClassNameTable.Value(new_bsp.class1 + ":" + new_bsp.name) = replacementIndex

if convert_magic then
    //convert the offset to the header of the BSP
    dim bw as new BinaryStream(tags(replacementIndex).data.bin)
    bw.LittleEndian = true
    bw.position = 0
    dim temp_offset_to_start as integer = bw.readint32
    temp_offset_to_start = temp_offset_to_start - old_magic
    temp_offset_to_start = temp_offset_to_start + new_magic
    bw.position = 0
    bw.writeint32(temp_offset_to_start)
    bw.close
end

//DOBC and deca problems:
//in the DOBC chunks, null the counts of the reflexives
//null the count of the deca reflexive
//prompt the user if they want these corrections

```

```

dim temp_S BSP as new SBSP_class_EX(tags(replacementIndex).data.bin, new_magic)
temp_S BSP.read
if (temp_S BSP.dobc_refs.ubound > -1) or (temp_S BSP.header.decal_reflexive.count > 0) then
    Dim d as New MessageDialog //declare the MessageDialog object
    Dim b as MessageDialogButton //for handling the result
    d.icon=MessageDialog.GraphicCaution //display warning icon
    d.ActionButton.Caption="Continue"
    d.CancelButton.Visible= True //show the Cancel button
    d.AlternateActionButton.Visible= True //show the "Don't Save" button
    d.AlternateActionButton.Caption="Remove References"
    d.Message="Warning: This SBSP tag has implied references"
    d.Explanation="DOBC and deca references that have dependencies in the SCNR can lead to unplayable map
files." + _
    " Would you like to remove these references?"
    b=d.ShowModal //display the dialog
    Select Case b //determine which button was pressed.
    Case d.ActionButton
        //do nothing
    Case d.AlternateActionButton
        //remove references
        dim bw as new BinaryStream(tags(replacementIndex).data.bin)
        bw.LittleEndian = true

        //dobc first
        for i as integer = 0 to temp_S BSP.dobc_refs.ubound
            //unknown reflexive 1
            for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
                if tags(replacementIndex).data.reflexives(j).offset = _
                    temp_S BSP.dobc_refs(i).unknown_reflexive_1.offset then
                    tags(replacementIndex).data.reflexives(j).count = 0
                    exit for j
            end
        next
        bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_1.offset
        bw.WriteUInt32(0)

        //unknown reflexive 2
        for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
            if tags(replacementIndex).data.reflexives(j).offset = _
                temp_S BSP.dobc_refs(i).unknown_reflexive_2.offset then
                tags(replacementIndex).data.reflexives(j).count = 0
                exit for j
        end
    next
    bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_2.offset
    bw.WriteUInt32(0)

    //unknown reflexive 3
    for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
        if tags(replacementIndex).data.reflexives(j).offset = _
            temp_S BSP.dobc_refs(i).unknown_reflexive_3.offset then
            tags(replacementIndex).data.reflexives(j).count = 0
            exit for j
        end
    end
end

```

```

next
bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_3.offset
bw.WriteUInt32(0)

//unknown reflexive 4
for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(j).offset = _
        temp_S BSP.dobc_refs(i).unknown_reflexive_4.offset then
            tags(replacementIndex).data.reflexives(j).count = 0
            exit for j
        end
    end
next
bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_4.offset
bw.WriteUInt32(0)

```

```

next
//deca next
for i as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(i).offset = _
        temp_S BSP.header.decal_reflexive.offset then
            tags(replacementIndex).data.reflexives.remove(i)
            exit for i
        end
    end
next
bw.Position = temp_S BSP.header.decal_reflexive.offset
bw.WriteUInt32(0)
bw.WriteUInt32(0)
bw.close

```

Case d.CancelButton

return false//cancel the process

End select

end

```

//add all the dependent tags
dim extra_data(-1) as meta
for i as integer = 1 to UBound(new_meta.metas)
    dim temp_meta as new meta
    temp_meta = new_meta.metas(i).copy
    extra_data.Append temp_meta
next
dim temp_bool as boolean
temp_bool = addTags(extra_data, replacementindex, no_duplicates)

```

return temp\_bool

End Function

## Map.replaceSBSP:

```

Function replaceSBSP(replacementIndex as integer, new_meta as meta) As boolean
    if not me.expanded then return false//can't replace a tag if the map isn't expanded
    if replacementIndex < 0 OR replacementIndex > UBound(tags) then
        //bad index
        return false
    end
end

```

```

if tags(replacementindex).class1 <> "psbs" then return false//big time error
if new_meta.class1 <> "psbs" then return false//can't use a non-sbsp meta pack
if not new_meta.expanded then return false

dim bsp_data_index as integer = -1
for i as integer = 0 to UBound(bsp_data)
    if tags(replacementindex).tagID = bsp_data(i).tagID then
        bsp_data_index = i
        exit for i
    end
next
if bsp_data_index = -1 then return false//problem here as well

dim new_bsp as new meta
new_bsp = new_meta.copy

dim convert_magic as boolean = false

//check to see whether or not to convert BSP magic
//the problem comes from extracting BSP info I think. something is going wrong with either the construction
//of the xml file, or in it's subsequent implementation
//SBSPs really are a pain in the ass
#if DebugBuild
    if (new_bsp.map_version = 6 AND me.Header.version <> 6) _
        OR (new_bsp.map_version <> 6 AND me.Header.version = 6) then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Convert"
        d.CancelButton.Visible= True //show the Cancel button
        d.AlternateActionButton.Visible= True //show the "Don't Save" button
        d.AlternateActionButton.Caption="Don't Convert"
        d.Message="Warning: This SBSP tag has an incompatible magic value"
        d.Explanation="Would you like to convert the magic value? (This may partially corrupt the tag)"
        b=d.ShowModal //display the dialog
        Select Case b //determine which button was pressed.
        Case d.ActionButton
            convert_magic = true
        Case d.AlternateActionButton
            convert_magic = false
        Case d.CancelButton
            return false//cancel the process
        End select
    end
#else
    if (new_bsp.map_version = 6 AND me.Header.version <> 6) _
        OR (new_bsp.map_version <> 6 AND me.Header.version = 6) then
        errorbox("Error: this BSP is incompatible with the target map")
        return false
    end
#endif

dim new_magic as integer
if convert_magic then

```

```

    new_bsp.map_version = me.Header.version
end
if new_bsp.map_version = 6 then
    new_magic = &h4D610000 - new_bsp.data.size
else
    new_magic = &h41B40000 - new_bsp.data.size
end
dim new_size as integer = new_bsp.data.size

//ok, now we do the deed
bsp_data(bsp_data_index).magic = new_magic
bsp_data(bsp_data_index).size = new_size
new_bsp.tagID = tags(replacementIndex).tagID
tags(replacementIndex) = new_bsp
tagClassNameTable.Value(new_bsp.class1 + ":" + new_bsp.name) = replacementIndex

//DOBC and deca problems:
//in the DOBC chunks, null the counts of the reflexives
//null the count of the deca reflexive
//prompt the user if they want these corrections
dim temp_S BSP as new SBSP_class_EX(tags(replacementIndex).data.bin, new_magic)
temp_S BSP.read
if (temp_S BSP.dobc_refs.ubound > -1) or (temp_S BSP.header.decal_reflexive.count > 0) then
    Dim d as New MessageDialog //declare the MessageDialog object
    Dim b as MessageDialogButton //for handling the result
    d.icon=MessageDialog.GraphicCaution //display warning icon
    d.ActionButton.Caption="Continue"
    d.CancelButton.Visible= True //show the Cancel button
    d.AlternateActionButton.Visible= True //show the "Don't Save" button
    d.AlternateActionButton.Caption="Remove References"
    d.Message="Warning: This SBSP tag has implied references"
    d.Explanation="DOBC and deca references that have dependencies in the SCNR can lead to unplayable map
files." + _
    " Would you like to remove these references?"
    b=d.ShowModal //display the dialog
    Select Case b //determine which button was pressed.
    Case d.ActionButton
        //do nothing
    Case d.AlternateActionButton
        //remove references
        dim bw as new BinaryStream(tags(replacementIndex).data.bin)
        bw.LittleEndian = true

        //dobc first
        for i as integer = 0 to temp_S BSP.dobc_refs.ubound
            //unknown reflexive 1
            for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
                if tags(replacementIndex).data.reflexives(j).offset = _
                    temp_S BSP.dobc_refs(i).unknown_reflexive_1.offset then
                    tags(replacementIndex).data.reflexives(j).count = 0
                    exit for j
                end
            next
            bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_1.offset
        
```

```

bw.WriteUInt32(0)

//unknown reflexive 2
for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(j).offset = _
        temp_S BSP.dobc_refs(i).unknown_reflexive_2.offset then
            tags(replacementIndex).data.reflexives(j).count = 0
        exit for j
    end
next
bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_2.offset
bw.WriteUInt32(0)

//unknown reflexive 3
for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(j).offset = _
        temp_S BSP.dobc_refs(i).unknown_reflexive_3.offset then
            tags(replacementIndex).data.reflexives(j).count = 0
        exit for j
    end
next
bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_3.offset
bw.WriteUInt32(0)

//unknown reflexive 4
for j as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(j).offset = _
        temp_S BSP.dobc_refs(i).unknown_reflexive_4.offset then
            tags(replacementIndex).data.reflexives(j).count = 0
        exit for j
    end
next
bw.Position = temp_S BSP.dobc_refs(i).unknown_reflexive_4.offset
bw.WriteUInt32(0)

next
//deca next
for i as integer = 0 to tags(replacementIndex).data.reflexives.ubound
    if tags(replacementIndex).data.reflexives(i).offset = _
        temp_S BSP.header.decal_reflexive.offset then
            tags(replacementIndex).data.reflexives.remove(i)
        exit for i
    end
next
bw.Position = temp_S BSP.header.decal_reflexive.offset
bw.WriteUInt32(0)
bw.WriteUInt32(0)
bw.close

Case d.CancelButton
    return false//cancel the process
End select
end

```



```
    return true
End Function
```

## Map.write:

```
Function write(f as folderitem, w as map_rebuild_window) As boolean
    //this is a simple rebuild method that only rebuilds for added tags and doesn't mess with the underlying
    structure
    'Write dummy header
    'write bsp zeros
    'write vert model section, store vert offset
    'write ind model section, store ind offset, store model data size
    'write internalized bitmaps
    'write internalized sounds
    'write new index without offsets
    'write tags
    'rewrite header with map size, meta size, offset to index
    'rewrite header with tag offsets
    'rewrite index with base tag
    'rewrite bsp_data

    if not expanded then return false //catch bad calls to the write method

    fs = f
    dim bw as BinaryStream = f.CreateBinaryFile(H1_Filetypes.HaloMapFile)
    bw.LittleEndian = true

    dim progress as integer = 0

    w.tick("Writing header", progress)
    Header.write(bw)

    //header accounts for roughly 5%
    progress = 5

    //write the bsp(s) as zeros for now
    //bsp zero writing are allocated 10%, total is at 15%
    for i as integer = 0 to UBound(bsp_data)
        bsp_data(i).offset = bw.Position
        if tagIDTable.HasKey(bsp_data(i).tagID) then
            dim size as integer = tags(tagIDTable.Value(bsp_data(i).tagID)).data.bin.size
            dim d as double = i
            dim num as double = (d*10) + 1
            dim denom as double = ubound(bsp_data) + 1
            dim add as integer = round(num/denom)
            w.tick("Writing BSP padding", progress + add)
            for j as integer = 1 to size
                bw.WriteByte(0)
            next
        else
            break
        end
    next
    progress = 15
```

```

//check for bitmaps
dim class_index as integer = -1
class_index = -1
for i as integer = 0 to UBound(meta_class_folders.child)
    if meta_class_folders.child(i).name.mid(1,4) = "bitm" then
        class_index = i
        exit for i
    end
next
if class_index >= 0 then
    for i as integer = 0 to UBound(meta_class_folders.child(class_index).item)
        //tags(meta_class_folders.child(class_index).item(i))
        //write bitmap data
        //sound data is allocated 5%, total is 20%
        if ubound(tags(meta_class_folders.child(class_index).item(i)).raw_data) >= 0 then
            dim d as double = i
            dim num as double = (d*5) + 1
            dim denom as double = ubound(meta_class_folders.child(class_index).item) + 1
            dim add as integer = round(num/denom)
            w.tick("Writing internalized bitmap data", progress + add)
        end
        for j as integer = 0 to UBound(tags(meta_class_folders.child(class_index).item(i)).raw_data)
            dim temp_offset as integer = bw.Position
            dim br as new BinaryStream(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).bin)
            br.LittleEndian = true
            while not br.EOF
                bw.WriteByte(br.ReadByte)
            wend
            br.close
            tags(meta_class_folders.child(class_index).item(i)).raw_data(j).offset = temp_offset
        next
    next
end
progress = 20

//check for sounds
class_index = -1
for i as integer = 0 to UBound(meta_class_folders.child)
    if meta_class_folders.child(i).name.mid(1,4) = "snd!" then
        class_index = i
        exit for i
    end
next
if class_index >= 0 then
    for i as integer = 0 to UBound(meta_class_folders.child(class_index).item)
        //tags(meta_class_folders.child(class_index).item(i))
        //write sound data
        //sound data is allocated 5%, total is 25%
        if ubound(tags(meta_class_folders.child(class_index).item(i)).raw_data) >= 0 then
            dim d as double = i
            dim num as double = (d*5) + 1
            dim denom as double = ubound(meta_class_folders.child(class_index).item) + 1
            dim add as integer = round(num/denom)
            w.tick("Writing internalized sound data", progress + add)
        end
    next
end

```

```

end
for j as integer = 0 to UBound(tags(meta_class_folders.child(class_index).item(i)).raw_data)
    dim temp_offset as integer = bw.Position
    dim br as new BinaryStream(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).bin)
    br.LittleEndian = true
    while not br.EOF
        bw.WriteByte(br.ReadByte)
    wend
    br.close
    tags(meta_class_folders.child(class_index).item(i)).raw_data(j).offset = temp_offset
next
next
end
//incase no internal bitmap or sound data
progress = 25

//check for mod2
class_index = -1
dim vert_count as integer = 0
dim vert_offset as integer = bw.Position
dim ind_count as integer = 0
dim ind_offset as integer
dim model_raw_size as integer
for i as integer = 0 to UBound(meta_class_folders.child)
    if meta_class_folders.child(i).name.mid(1,4) = "mod2" then
        class_index = i
        exit for i
    end
next
if class_index >= 0 then
    for i as integer = 0 to UBound(meta_class_folders.child(class_index).item)
        //tags(meta_class_folders.child(class_index).item(i))
        //write vert data
        //vert data is allocated 5%, total is 30%
        dim d as double = i
        dim num as double = (d*5) + 1
        dim denom as double = ubound(meta_class_folders.child(class_index).item) + 1
        dim add as integer = round(num/denom)
        w.tick("Writing model vertex data", progress + add)
        for j as integer = 0 to UBound(tags(meta_class_folders.child(class_index).item(i)).raw_data)
            dim temp_str() as string = split(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).str_info,
            ":")
            if temp_str(UBound(temp_str)) = "vert" then

                dim temp_offset as integer = bw.Position
                dim br as new BinaryStream(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).bin)
                br.LittleEndian = true
                while not br.EOF
                    bw.WriteByte(br.ReadByte)
                wend
                br.close
                tags(meta_class_folders.child(class_index).item(i)).raw_data(j).offset = temp_offset
                vert_count = vert_count + 1
            end
        next
    next
end

```

```

    next
next
ind_offset = bw.Position - vert_offset
progress = 30

for i as integer = 0 to UBound(meta_class_folders.child(class_index).item)
    //write ind data
    //ind data is allocated 5%, total is 35%
    dim d as double = i
    dim num as double = (d*5) + 1
    dim denom as double = ubound(meta_class_folders.child(class_index).item) + 1
    dim add as integer = round(num/denom)
    w.tick("Writing model index data", progress + add)
    for j as integer = 0 to UBound(tags(meta_class_folders.child(class_index).item(i)).raw_data)
        dim temp_str() as string = split(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).str_info,
            ":")
        if temp_str(UBound(temp_str)) = "ind" then

            dim temp_offset as integer = bw.Position
            dim br as new BinaryStream(tags(meta_class_folders.child(class_index).item(i)).raw_data(j).bin)
            br.LittleEndian = true
            while not br.EOF
                bw.WriteByte(br.ReadByte)
            wend
            br.close
            tags(meta_class_folders.child(class_index).item(i)).raw_data(j).offset = temp_offset
            ind_count = ind_count + 1
        end
    next
next
model_raw_size = bw.Position - vert_offset
else
    //this should mean that there were no mod2 files
    break
end
model_raw_size = bw.Position - vert_offset
progress = 35

//new magic calculation
dim index_offset as integer = bw.Position
dim map_magic as integer = Index_Header.index_magic
map_magic = map_magic - (index_offset + 36)
if (Header.version <> 5) then
    map_magic = map_magic - 4
end

Index_Header.ind_count = ind_count
Index_Header.ind_offset = ind_offset
Index_Header.vert_count = vert_count
Index_Header.vert_offset = vert_offset
Index_Header.ModelRawDataSize = model_raw_size
Index_Header.tagcount = UBound(tags)+1 //since arrays are zero based

Index_Header.write(bw, header.version <> 5)

```

```

dim temp_ID1 as uint32 = 0
dim temp_ID2 as uint32 = 57716
dim first_scnr as boolean = true
dim basetagID as uint32 = 0
dim bsp_inds(-1) as integer
for i as integer = 0 to UBound(bsp_data)
    if tagIDTable.HasKey(bsp_data(i).tagID) then
        bsp_inds.append tagIDTable.Value(bsp_data(i).tagID)
    else
        break
    end
next
tagIDTable = new Dictionary
for i as integer = 0 to UBound(tags)
    dim current_ID as uint32 = Bitwise.BitOr(Bitwise.ShiftLeft(temp_ID2, 16), temp_ID1)
    tags(i).tagID = current_ID
    if tags(i).class1 = "rncs" AND ((i = base_scnr) or (base_scnr = -1 and first_scnr)) then
        basetagID = current_ID
        first_scnr = false
    end
    //index data is allocated 10%, total is 45%
    dim d as double = i
    dim num as double = (d*10) + 1
    dim denom as double = ubound(tags) + 1
    dim add as integer = round(num/denom)
    w.tick("Writing tag index data", progress + add)
    tags(i).write(bw, map_magic)
    tagIDTable.Value(current_ID) = i
    temp_ID1 = temp_ID1 + 1
    temp_ID2 = temp_ID2 + 1
next
//correct bsp_data tagID info
for i as integer = 0 to UBound(bsp_data)
    bsp_data(i).tagID = tags(bsp_inds(i)).tagID
next

progress = 45

for i as integer = 0 to UBound(tags)
    tags(i).nameoffset = bw.Position
    //string data is allocated 5%, total is 50%
    dim d as double = i
    dim num as double = (d*5) + 1
    dim denom as double = ubound(tags) + 1
    dim add as integer = round(num/denom)
    w.tick("Writing tag name data", progress + add)
    bw.Write(tags(i).name + chr(0))
next

progress = 50

//correct bsp_data name_offset info
for i as integer = 0 to UBound(bsp_data)

```

```

if tagIDTable.HasKey(bsp_data(i).tagID) then
    bsp_data(i).dep_name_offset = tags(tagIDTable.Value(bsp_data(i).tagID)).nameoffset
end
next

for i as integer = 0 to UBound(tags)
    //tag data is allocated 45%, total is 95%
    dim d as double = i
    dim num as double = (d*45) + 1
    dim denom as double = ubound(tags) + 1
    dim add as integer = round(num/denom)
    w.tick("Writing tag data", progress + add)
    if tags(i).class1 <> "psbs" then
        tags(i).offset = bw.Position
        writeTag(bw, tags(i), map_magic, map_magic)
    else
        dim scnr_bsp as boolean = false
        for j as integer = 0 to UBound(bsp_data)
            if tags(i).tagID = bsp_data(j).tagID then
                scnr_bsp = true
                exit for j
            end
        next
        if not scnr_bsp then
            tags(i).offset = bw.Position
            writeTag(bw, tags(i), map_magic, map_magic)
        end
    end

    if tags(i).tagID = basetagID then
        //fix the scenario listing for bsp info
        dim temp_offset as integer = bw.position
        bw.close //necessary because windows doesn't like multiple streams attached to the same file
        dim br as BinaryStream = f.openasbinaryfile
        br.LittleEndian = true
        br.Position = tags(i).offset + &h5A4

        dim count_offset as integer = br.Position
        dim count as integer = br.ReadInt32
        dim offset as integer = br.ReadInt32 - map_magic
        br.close
        bw = f.openasbinaryfile(true)
        bw.littleendian = true
        bw.position = temp_offset

        if count <> UBound(bsp_data) + 1 then
            bw.Position = count_offset
            count = UBound(bsp_data) + 1
            bw.WriteUInt32(count)
        end
        bw.Position = offset
        for j as integer = 0 to UBound(bsp_data)
            bsp_data(j).write(bw, map_magic)
        next
    end

```

```

        bw.Position = temp_offset
    end
next

progress = 95

dim length as integer = bw.Position

//rewrite header
bw.Position = 0
Header.decomp_len = length
Header.TagIndexOffset = index_offset
Header.TagIndexMetaLength = length - index_offset
Header.write(bw)

//rewrite index
bw.Position = index_offset
Index_Header.BaseTag = basetagID
Index_Header.write(bw, Header.version <> 5)
for i as integer = 0 to UBound(tags)
    tags(i).write(bw, map_magic)
next

//rewrite bsp datas to include updated
for i as integer = 0 to UBound(bsp_data)
    bw.Position = bsp_data(i).offset
    //bsp data is allocated 5%, total is 100%
    dim d as double = i
    dim num as double = (d*5) + 1
    dim denom as double = ubound(bsp_data) + 1
    dim add as integer = round(num/denom)
    w.tick("Writing bsp index data", progress + add)
    if tagIDTable.HasKey(bsp_data(i).tagID) then
        writeTag(bw, tags(tagIDTable.Value(bsp_data(i).tagID)), bsp_data(i).magic, map_magic, true)
    else
        break
    end
next

Progress = 100
w.tick("Finishing up", progress)

bw.Close
Return true
End Function

```

### Map.writeTag:

```

Sub writeTag(byref bw as binaryStream, data as meta, local_magic as integer, map_magic as integer, no_local as
boolean = false)
    //no local means no local offset

    if data.CE_flag = 1 then return //quit it

    dim offset as integer = bw.Position

```

```

bw.LittleEndian = true

dim br as new BinaryStream(data.data.bin)
br.LittleEndian = true

for i as integer = 1 to data.data.size
    bw.WriteByte(br.ReadByte)
next
dim end_pos as integer = bw.Position

for i as integer = 0 to UBound(data.data.reflexives)
    bw.Position = offset + data.data.reflexives(i).offset
    bw.WriteUInt32(data.data.reflexives(i).count)
    if no_local then
        bw.WriteUInt32(data.data.reflexives(i).translation + local_magic)
    else
        bw.WriteUInt32(data.data.reflexives(i).translation + offset + local_magic)
    end
next

for i as integer = 0 to UBound(data.data.loneIDs)
    bw.Position = offset + data.data.loneIDs(i).offset
    dim tagclass as string = data.data.loneIDs(i).tag_class
    dim filename as string = data.data.loneIDs(i).tag_name
    if tagClassNameTable.HasKey(tagclass.Mid(1,4)+":"+filename) then
        bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass+":"+filename)).tagID)
    else
        bw.WriteInt32(-1)
    end
next

for i as integer = 0 to UBound(data.data.dependencies)
    dim location as integer = data.data.dependencies(i).offset
    dim tagclass as string = data.data.dependencies(i).tag_class
    dim filename as string = data.data.dependencies(i).tag_name
    if tagClassNameTable.HasKey(tagclass.Mid(1,4)+":"+filename) then
        bw.Position = offset + location
        bw.Write(tags(tagClassNameTable.Value(tagclass+":"+filename)).class1)
        bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass+":"+filename)).nameoffset + map_magic)
        bw.WriteUInt32(0)
        bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass+":"+filename)).tagID)
    else
        bw.Position = offset + location + 12
        bw.WriteInt32(-1)
    end
next
//finished writing meta data
bw.Position = end_pos

//extra options for expansion in order to keep track of raw data
select case data.class1

case "2dom"
    //extract all the raw data for vert and inds

```



```

bw.close
dim m as new mod2_class(fs, data.offset, map_magic)
m.read
bw = fs.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = end_pos
for i as integer = 0 to UBound(data.raw_data)
    dim temp_str() as string = split(data.raw_data(i).str_info,":")
    if temp_str.ubound = 5 then //should be six long
        //"mod2:geometry:00:part:00:vert"
        //"mod2:geometry:00:part:00:ind"
        dim geo_ind as integer = val(temp_str(2))
        dim part_ind as integer = val(temp_str(4))
        dim type as string = temp_str(5)
        dim new_offset as integer = data.raw_data(i).offset
        dim success as Boolean = false
        dim current_pos as integer = bw.Position
        select case type
            case "vert"
                success = m.update_offset(bw, geo_ind, part_ind, new_offset-Index_Header.vert_offset, 1)
            case "ind"
                success = m.update_offset(bw, geo_ind, part_ind, new_offset-(Index_Header.vert_offset +
                    Index_Header.ind_offset), 2)
        end select
        bw.Position = current_pos
        if not success then
            break
        end
    end
next

case "!dns"
//extract the raw data for sounds if internalized
bw.close
dim s as new snd_class_RW(fs, fs, data.offset, map_magic)
s.read
bw = fs.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = end_pos

for i as integer = 0 to UBound(data.raw_data)
    dim temp_str() as string = split(data.raw_data(i).str_info,":")
    if temp_str.ubound = 4 then //should be five long
        //"snd!:pitch:00:permutation:00"
        dim pitch_ind as integer = val(temp_str(2))
        dim perm_ind as integer = val(temp_str(4))
        dim new_offset as integer = data.raw_data(i).offset
        dim success as Boolean = false
        dim current_pos as integer = bw.Position
        success = s.update_offset(bw, pitch_ind, perm_ind, new_offset, True)
        bw.Position = current_pos
        if not success then
            break
        end
    end

```

```

        end
    next

case "mtib"
    //extract the raw data for bitmaps if internalized
    bw.close
    dim b as new bitmap_class_RW(fs, fs, data.offset, map_magic)
    b.read
    bw = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = end_pos

    for i as integer = 0 to UBound(data.raw_data)
        dim temp_str() as string = split(data.raw_data(i).str_info,":")
        if temp_str.ubound = 2 then //should be three long
            //"bitm:image_info:00"
            dim image_ind as integer = val(temp_str(2))
            dim new_offset as integer = data.raw_data(i).offset
            dim success as Boolean = false
            dim current_pos as integer = bw.Position
            success = b.update_offset(bw, image_ind, new_offset, True)
            bw.Position = current_pos
            if not success then
                break
            end
        end
    next

end select

bw.Position = end_pos
End Sub

```

### Map.write\_data:

```

Sub write_data(tag_ind as integer)
    dim f as FolderItem = GetSaveFolderItem("", "file data")
    dim input_stream as new BinaryStream( tags(tag_ind).data.bin)
    dim output_stream as BinaryStream = f.CreateBinaryFile("")
    while not input_stream.EOF
        output_stream.WriteByte(input_stream.ReadByte)
    wend
    input_stream.close
    output_stream.close

End Sub

base_scnr As Integer

bitmaps_f As folderItem

bsp_data(-1) As bsp_scenario_data

cTagTypes As dictionary

expanded As boolean

```

fs As folderItem

Header As Map\_header

Index\_Header As Map\_index\_Header

magic As Integer

meta\_class\_folders As vFolder

meta\_name\_folders As vFolder

sounds\_f As folderItem

tagClassNameTable As dictionary

tagIDTable As dictionary

tags(-1) As meta

### Map Note: old xml stuff

old xml stuff

tag expansion:

dim xdoc as new XmlDocument

xdoc.AppendChild(xdoc.CreateComment(" Eschaton: Meta Structure File "))

dim xml\_results as XmlNode

xml\_results = xdoc.AppendChild(xdoc.CreateElement("Results"))

dim xml\_map as XmlNode

xml\_map = xml\_results.AppendChild(xdoc.CreateElement("Map"))

xml\_map.AppendChild(xdoc.CreateTextNode(header.name))

dim xml\_tag as XmlNode

xml\_tag = xml\_results.AppendChild(xdoc.CreateElement("Tag"))

dim this\_class\_string as string = tags(index).class1 + tags(index).class2 + tags(index).class3

this\_class\_string = this\_class\_string.ReplaceAll(chr(&hFF), "F")

xml\_tag.AppendChild(xdoc.CreateTextNode(this\_class\_string))

dim xml\_filename as XmlNode

xml\_filename = xml\_results.AppendChild(xdoc.CreateElement("Filename"))

xml\_filename.AppendChild(xdoc.CreateTextNode(tags(index).name))

dim dep\_list(-1) as UInt32

for i as integer = 0 to UBound(temp\_data)

dim tempint as integer = temp\_data(i)

if (not(tempint = 0)) AND (not(tempint = &hFFFFFF)) AND (not(tempint = &hCACACACA)) then

dim tempintmagic as integer = tempint - local\_magic

dim reflexive\_count as integer

dim intTranslation as integer

if isReflexiveOffset(tempintmagic, data\_start, data\_size) AND (data\_start > 0) AND (data\_size > 0) then

if (i >= 1) AND (tags(index).class1 <> "psbs" OR allow\_bsp\_reflexives) then

//reflexive

dim zero\_test as integer = -1

if i+1 <= UBound(temp\_data) then

zero\_test = temp\_data(i+1)

```

end
if zero_test = 0 then //reflexives need to have a chunk count and a set of zeros following them
dim ref as xmlnode
dim location as xmlnode
dim translation as xmlnode
dim chunkcount as xmlnode
ref = xml_results.AppendChild(xdoc.CreateElement("Reflexive"))
location = ref.AppendChild(xdoc.CreateElement("Location"))
location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
intTranslation = tempintmagic - tags(index).offset
translation = ref.AppendChild(xdoc.CreateElement("Translation"))
translation.AppendChild(xdoc.CreateTextNode("0x" + hex(intTranslation) ))
chunkcount = ref.AppendChild(xdoc.CreateElement("ChunkCount"))
chunkcount.AppendChild(xdoc.CreateTextNode(str(temp_data(i-1))))

end
end
else

if(tagIDTable.HasKey(temp_data(i))) then
//either a loneID or a dependency
dim item as new meta
item = tags(tagIDTable.Value(temp_data(i)))
dep_list.Append(temp_data(i))

dim dep as xmlnode
dim location as xmlnode
dim tagclass as xmlnode
dim filename as xmlnode
if (i >= 3) AND isTag(temp_data(i-3)) then
//dependency
dep = xml_results.AppendChild(xdoc.CreateElement("Dependency"))
location = dep.AppendChild(xdoc.CreateElement("Location"))
location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
tagclass = dep.AppendChild(xdoc.CreateElement("Tagclass"))
dim class_string as string = item.class1 + item.class2 + item.class3
class_string = class_string.ReplaceAll(chr(&hFF), "F")
tagclass.AppendChild(xdoc.CreateTextNode(class_string))
filename = dep.AppendChild(xdoc.CreateElement("Filename"))
filename.AppendChild(xdoc.CreateTextNode(item.name))

else
if tags(index).class1 <> "psbs" OR true then //was originally checking if sbasp loneID updating was breaking
dep = xml_results.AppendChild(xdoc.CreateElement("LoneID"))
location = dep.AppendChild(xdoc.CreateElement("Location"))
location.AppendChild(xdoc.CreateTextNode("0x" + hex((i)*4) ))
tagclass = dep.AppendChild(xdoc.CreateElement("Tagclass"))
dim class_string as string = item.class1 + item.class2 + item.class3
class_string = class_string.ReplaceAll(chr(&hFF), "F")
tagclass.AppendChild(xdoc.CreateTextNode(class_string))
filename = dep.AppendChild(xdoc.CreateElement("Filename"))
filename.AppendChild(xdoc.CreateTextNode(item.name))

end

```

end  
end

end

end  
next

tag writing:

```
dim xdoc as XmlDocument = data.data.xdoc
for i as integer = 0 to xdoc.DocumentElement.ChildCount-1
dim node as XmlNode = xdoc.DocumentElement.Child(i)
select case node.Name
case "Map"
//no use
case "Tag"
//no use
case "Filename"
//no use
case "Reflexive"
dim location as integer = -1
dim translation as integer = -1
dim chunkcount as integer = -1
try
location = val("&h" + node.Child(0).FirstChild.Value.Mid(3))
catch err as NilObjectException
location = -1
end try
try
translation = val("&h" + node.Child(1).FirstChild.Value.Mid(3))
catch err as NilObjectException
translation = -1
end try
try
chunkcount = val(node.Child(2).FirstChild.Value)
catch err as NilObjectException
chunkcount = -1
end try
if location <> -1 AND translation <> -1 and chunkcount <> -1 then
bw.Position = offset + location - 4
bw.WriteUInt32(chunkcount)
if no_local then
bw.WriteUInt32(translation + local_magic)
else
bw.WriteUInt32(translation + offset + local_magic)
end
end
case "LoneID"
dim location as integer = -1
dim tagclass as string = "NULL"
dim filename as string = "NULL"
try
location = val("&h" + node.Child(0).FirstChild.Value.Mid(3))
catch err as NilObjectException
```

```

location = -1
end try
try
tagclass = node.Child(1).FirstChild.Value
catch err as NilObjectException
tagclass = "NULL"
end try
try
filename = node.Child(2).FirstChild.Value
catch err as NilObjectException
filename = "NULL"
end try
if location <> -1 AND tagclass <> "NULL" and filename <> "NULL" then
bw.Position = offset + location
if tagClassNameTable.HasKey(tagclass.Mid(1,4)+":"+filename) then
bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass.Mid(1,4)+":"+filename)).tagID)
else
bw.WriteInt32(-1)
end
end
case "Dependency"
dim location as integer = -1
dim tagclass as string = "NULL"
dim filename as string = "NULL"
try
location = val("&h" + node.Child(0).FirstChild.Value.Mid(3))
catch err as NilObjectException
location = -1
end try
try
tagclass = node.Child(1).FirstChild.Value
catch err as NilObjectException
tagclass = "NULL"
end try
try
filename = node.Child(2).FirstChild.Value
catch err as NilObjectException
filename = "NULL"
end try
if location <> -1 AND tagclass <> "NULL" and filename <> "NULL" then
if tagClassNameTable.HasKey(tagclass.Mid(1,4)+":"+filename) then
bw.Position = offset + location - 12
bw.Write(tags(tagClassNameTable.Value(tagclass.Mid(1,4)+":"+filename)).class1)
bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass.Mid(1,4)+":"+filename)).nameoffset + map_magic)
bw.WriteUInt32(0)
bw.WriteUInt32(tags(tagClassNameTable.Value(tagclass.Mid(1,4)+":"+filename)).tagID)
else
bw.Position = offset + location + 12
bw.WriteInt32(-1)
end
end
end select
next

```

## Map Note: Rebuilding Methods

### Rebuilding Methods

#### Method 1:

##### Simple Tag Import

All that's necessary for this is to add tags to the index at the end of the map

Write dummy header

write bsp raw

write internalized bitmaps

write internalized sounds

write vert model section, store vert offset

write ind model section, store ind offset, store model data size

write new index without offsets

write tags

rewrite header with map size, meta size, offset to index

rewrite header with tag offsets

#### Method 2:

//doesn't make sense with current design paradigm

//although tag reordering operates on a similar design

##### Full Rebuild

use HMT like interface

scnr

matg: globals\globals

tagc: ui\ui\_tags\_loaded\_all\_scenario\_types

bitm: ui\shell\bitmaps\background

ustr: ui\shell\strings\loading

ustr: ui\shell\main\_menu\mp\_map\_list

bitm: ui\shell\bitmaps\trouble\_brewing

snd!: sound\sfx\ui\cursor

snd!: sound\sfx\ui\forward

snd!: sound\sfx\ui\back

tagc: ui\ui\_tags\_loaded\_multiplayer\_scenario\_type | ui\ui\_tags\_loaded\_solo\_scenario\_type

sbsp

in that order add tags to the index using recursive tag building (huh, sbsp is included in the scnr tag)

then make sure to add any unused tags as well

this should allow for sp->mp rebuilding

## Map Note: REMEMBER FOR WRITING TAGS

### REMEMBER FOR WRITING TAGS

dependencies NEED to include the string offset as well id and class1 data

otherwise it won't be as powerful as the 16byte swap

End Class

Class Map\_Header

## Map\_Header.Constructor:

Sub Constructor()

redim demo\_junk(-1)

```
redim reserved1(-1)
redim reserved1(-1)
End Sub
```

### **Map\_Header.read:**

```
Function read(byref br as binaryStream) As boolean
    br.LittleEndian = true
```

```
    head_string = br.Read(4)
    version = br.ReadInt32
```

```
    if head_string <> "daeh" then
        br.Position = &h588
        version = br.readInt32
```

```
    if version = 6 then
```

```
        br.position = &h2c0
        head_string = br.read(4)
```

```
        br.position = &h5e8
        decomp_len = br.readint32
```

```
        br.position = &h5ec
        TagIndexOffset = br.readint32
```

```
        br.position = &h2c4
        TagIndexMetaLength = br.readint32
```

```
        br.position = &h58c
        name = br.Read(32)
```

```
        br.position = &h2c8
        BuildDate = br.read(32)
```

```
        br.position = &h02
        MapType = br.ReadInt16
```

```
        br.position = 100
        unknown4 = br.readint32
```

```
        br.position = &h5f0
        foot_string = br.read(4)
```

```
        redim demo_junk(-1)
        br.Position = 0
        for i as integer = 1 to (&h800)/4
            demo_junk.Append(br.ReadInt32)
        next
```

```
        return true
    else
        return false
    end
```



else

```
decomp_len = br.ReadInt32
unknown1 = br.ReadInt32
TagIndexOffset = br.ReadInt32
TagIndexMetaLength = br.ReadInt32
redim reserved1(-1)
reserved1.Append(br.ReadInt32)
reserved1.Append(br.ReadInt32)
name = br.Read(32)
BuildDate = br.Read(32)
MapType = br.ReadInt16
unknown3 = br.ReadInt16
unknown4 = br.ReadInt32
redim reserved2(-1)
for i as integer = 1 to &h794
    reserved2.Append(br.ReadInt32)
next
foot_string = br.Read(4)
```

return true

end

return false

End Function

### **Map\_Header.update:**

Sub update(byref bw as binaryStream, new\_len as integer, new\_index\_offset as integer, new\_meta\_length as integer)

bw.LittleEndian = true

decomp\_len = new\_len

TagIndexOffset = new\_index\_offset

TagIndexMetaLength = new\_meta\_length

if version = 6 then

dim position as integer = bw.Position

bw.position = &h5e8

bw.WriteInt32(decomp\_len)

bw.position = &h5ec

bw.WriteInt32(TagIndexOffset)

bw.position = &h2c4

bw.WriteInt32(TagIndexMetaLength)

else

bw.WriteInt32(decomp\_len)

bw.WriteInt32(TagIndexOffset)

bw.WriteInt32(TagIndexMetaLength)

end

End Sub

## Map\_Header.write:

```
Sub write(byref bw as binaryStream)
```

```
    bw.LittleEndian = true
```

```
    if version = 6 then
```

```
        dim position as integer = bw.Position
```

```
        for i as integer = 1 to (&h800)/4
```

```
            bw.WriteInt32(demo_junk(i-1))
```

```
        next
```

```
        //other stuff for writing of demo headerness
```

```
        bw.position = &h2c0
```

```
        bw.Write(head_string)
```

```
        bw.position = &h588
```

```
        bw.WriteInt32(version)
```

```
        bw.position = &h5e8
```

```
        bw.WriteInt32(decomp_len)
```

```
        bw.position = &h5ec
```

```
        bw.WriteInt32(TagIndexOffset)
```

```
        bw.position = &h2c4
```

```
        bw.WriteInt32(TagIndexMetaLength)
```

```
        bw.position = &h58c
```

```
        bw.Write(name)
```

```
        bw.position = &h2c8
```

```
        bw.Write(BuildDate)
```

```
        bw.position = &h02
```

```
        bw.WriteInt16(MapType)
```

```
        bw.position = 100
```

```
        bw.WriteInt32(unknown4)
```

```
        bw.position = &h5f0
```

```
        bw.Write(foot_string)
```

```
    else
```

```
        bw.Write(head_string)
```

```
        bw.WriteInt32(version)
```

```
        bw.WriteInt32(decomp_len)
```

```
        bw.WriteInt32(unknown1)
```

```
        bw.WriteInt32(TagIndexOffset)
```

```
        bw.WriteInt32(TagIndexMetaLength)
```

```
        bw.WriteInt32(reserved1(0))
```

```
        bw.WriteInt32(reserved1(1))
```

```
        bw.Write(name)
```

```
        bw.Write(BuildDate)
```

```

        bw.WriteInt16(MapType)
        bw.WriteInt16(unknown3)
        bw.WriteInt32(unknown4)
        for i as integer = 1 to &h794
            bw.WriteInt32(reserved2(i-1))
        next
        bw.Write(foot_string)
    end

```

```

        bw.Position = &h800
    End Sub

```

BuildDate As string

decomp\_len As Integer

demo\_junk(-1) As Integer

foot\_string As string

head\_string As string

MapType As Integer

name As string

reserved1(-1) As Integer

reserved2(-1) As Integer

TagIndexMetaLength As Integer

TagIndexOffset As Integer

unknown1 As Integer

unknown3 As Integer

unknown4 As Integer

version As Integer

### Map\_Header Note: HEADER\_EMPTY\_INTS

HEADER\_EMPTY\_INTS

//484

### Map\_Header Note: MapType

MapType

00 = single player

01 = multiplayer

02 = UI

End Class

Class Map\_Index\_Header

### Map\_Index\_Header.read:

```
Sub read(byref br as binaryStream, PC as boolean)
    br.LittleEndian = true

    index_magic = br.ReadInt32
    basetag = br.readUInt32
    unknown2 = br.ReadInt32
    tagcount = br.ReadInt32
    vert_count = br.ReadInt32
    vert_offset = br.ReadInt32 //offset to the start of the vertex data
    ind_count = br.ReadInt32
    ind_offset = br.ReadInt32 //offset to the start of the indices data relative to the start of the vert data
    if(PC) then
        ModelRawDataSize = br.ReadInt32 //total size of the model vert and ind data sections
    end
    footer = br.ReadInt32
End Sub
```

### Map\_Index\_Header.write:

```
Sub write(byref bw as binaryStream, PC as boolean = true)
    bw.WriteInt32(index_magic)
    bw.WriteUInt32(BaseTag)
    bw.WriteInt32(unknown2)
    bw.WriteInt32(tagcount)
    bw.WriteInt32(vert_count)
    bw.WriteInt32(vert_offset)
    bw.WriteInt32(ind_count)
    bw.WriteInt32(ind_offset)
    if (PC) then
        bw.WriteInt32(ModelRawDataSize)
    end
    bw.WriteInt32(footer)
End Sub
```

BaseTag As uint32

footer As Integer

index\_magic As Integer

ind\_count As Integer

ind\_offset As Integer

ModelRawDataSize As Integer

tagcount As Integer

unknown2 As Integer

vert\_count As Integer

vert\_offset As Integer

End Class

Class bsp\_scenario\_data

**bsp\_scenario\_data.read:**

Sub read(byref br as binaryStream, map\_magic as integer)

br.LittleEndian = true

offset = br.ReadInt32

size = br.ReadInt32

magic = br.ReadInt32

zero = br.ReadInt32

dep\_class = br.Read(4)

dep\_name\_offset = br.ReadInt32 - map\_magic

dep\_zero = br.ReadInt32

tagID = br.ReadUInt32

End Sub

**bsp\_scenario\_data.write:**

Sub write(byref bw as binaryStream, map\_magic as integer)

bw.LittleEndian = true

bw.WriteInt32(offset)

bw.WriteInt32(size)

bw.WriteInt32(magic)

bw.WriteInt32(zero)

bw.Write(dep\_class)

bw.WriteInt32(dep\_name\_offset + map\_magic)

bw.WriteInt32(dep\_zero)

bw.WriteUInt32(tagID)

End Sub

dep\_class As string

dep\_name\_offset As Integer

dep\_zero As Integer

magic As Integer

offset As Integer

size As Integer

tagID As uint32

zero As Integer

End Class

Class Meta

#### Meta.Constructor:

```
Sub Constructor()  
    data = new Meta_Expanded_Data  
    redim raw_data(-1)  
End Sub
```

#### Meta.copy:

```
Function copy() As meta  
    dim temp_meta as new meta  
  
    temp_meta.CE_flag = CE_flag  
    temp_meta.class1 = class1  
    temp_meta.class2 = class2  
    temp_meta.class3 = class3  
    temp_meta.data = data.copy  
    temp_meta.expanded = expanded  
    temp_meta.magic = magic  
    temp_meta.map_version = map_version  
    temp_meta.metasize = metasize  
    temp_meta.name = name  
    temp_meta.nameoffset = nameoffset  
    temp_meta.offset = offset  
    for i as integer = 0 to UBound(raw_data)  
        temp_meta.raw_data.Append raw_data(i).copy  
    next  
    temp_meta.tagID = tagID  
    temp_meta.zero = zero  
  
    return temp_meta  
End Function
```

#### Meta.generate\_xml:

```
Function generate_xml() As string  
    //this will output the meta data as an xml file converted to a string  
    //this uses v2.1 to create the xml file  
    if not expanded then return ""  
    dim xdoc as new XmlDocument  
    xdoc.AppendChild(xdoc.CreateComment(" Eschaton: Meta Structure File v2.1 "))  
    dim xml_results as XmlNode  
    xml_results = xdoc.AppendChild(xdoc.CreateElement("Results"))  
    dim xml_map as XmlNode  
    xml_map = xml_results.AppendChild(xdoc.CreateElement("Map"))  
    xml_map.setAttribute("name",map_name)  
    xml_map.setAttribute("version",str(map_version))  
    xml_map.setAttribute("magic","0x"+hex(magic))  
    dim xml_tag as XmlNode  
    xml_tag = xml_results.AppendChild(xdoc.CreateElement("Tag"))  
    xml_tag.setAttribute("class1", class1.ReplaceAll(chr(&hFF), "F"))  
    xml_tag.setAttribute("class2", class2.ReplaceAll(chr(&hFF), "F"))  
    xml_tag.setAttribute("class3", class3.ReplaceAll(chr(&hFF), "F"))  
    xml_tag.setAttribute("tagoffset", "0x" + hex(offset))  
    dim xml_filename as XmlNode  
    xml_filename = xml_results.AppendChild(xdoc.CreateElement("Filename"))
```

```

xml_filename.setAttribute("name",name)

for i as integer = 0 to data.reflexives.Ubound
    dim ref as XmlNode
    ref = xml_results.AppendChild(xdoc.CreateElement("Reflexive"))
    ref.SetAttribute("location", "0x" + hex(data.reflexives(i).offset))
    ref.SetAttribute("chunkcount", str(data.reflexives(i).count))
    ref.SetAttribute("translation", "0x" + hex(data.reflexives(i).translation))
next
for i as integer = 0 to data.dependencies.Ubound
    dim ref as XmlNode
    ref = xml_results.AppendChild(xdoc.CreateElement("Dependency"))
    ref.SetAttribute("location","0x" + hex(data.dependencies(i).offset))
    ref.SetAttribute("tagclass", data.dependencies(i).tag_class)
    ref.SetAttribute("filename", data.dependencies(i).tag_name)
next
for i as integer = 0 to data.loneIDs.Ubound
    dim ref as XmlNode
    ref = xml_results.AppendChild(xdoc.CreateElement("LoneID"))
    ref.SetAttribute("location","0x" + hex(data.loneIDs(i).offset))
    ref.SetAttribute("tagclass", data.loneIDs(i).tag_class)
    ref.SetAttribute("filename", data.loneIDs(i).tag_name)
next

dim output_str as string = xdoc.ToString
output_str = FileManip.xml_to_string(output_str)
return output_str
End Function

```

### Meta.read:

```

Sub read(byref br as binaryStream, magic as integer)
    br.LittleEndian = true

    class1 = br.Read(4)
    class2 = br.Read(4)
    class3 = br.Read(4)
    tagID = br.ReadUInt32
    nameoffset = br.ReadInt32 - magic

    //read the name string
    dim currentoffset as integer = br.Position
    dim peek as int8
    br.Position = nameoffset
    peek = br.ReadInt8
    br.Position = br.Position - 1
    name = ""
    while peek <> 0
        name = name + br.Read(1)
        peek = br.ReadInt8
        br.Position = br.Position - 1
    wend
    br.Position = currentoffset

    offset = br.ReadInt32

```

```

CE_flag = br.ReadInt32
if (offset > br.Position - 4) AND (CE_flag <> 1) then offset = offset - magic
if CE_flag = 1 AND class1 = "!dns" then
    offset = offset - magic
end
zero = br.ReadInt32
End Sub

```

### Meta.write:

```

Sub write(byref bw as binaryStream, magic as integer)
    bw.Write(class1)
    bw.Write(class2)
    bw.Write(class3)
    bw.WriteUInt32(tagID)
    bw.WriteInt32(nameoffset + magic)
    if (class1 = "psbs") then
        bw.WriteInt32(0)
    else
        if CE_flag = 1 then
            bw.WriteInt32(offset)
        else
            bw.WriteInt32(offset + magic)
        end
    end
    bw.WriteInt32(CE_flag)
    bw.WriteInt32(zero)
End Sub
CE_flag As Integer

```

class1 As string

class2 As string

class3 As string

data As Meta\_Expanded\_Data

expanded As boolean

### Meta.magic:

```

magic As Integer = -1
//local magic for dealing with expanded tags
map_name As string

```

### Meta.map\_version:

```

map_version As Integer = 0
//0 = unknown
//5 = xbox
//6 = demo
//7 = pc
//609 = CE (0x261)

```



```

metasize As Integer

name As string

nameoffset As integer

offset As Integer

raw_data(-1) As meta_raw

tagID As uint32

zero As Integer

End Class
Class Meta_Expanded_Data

```

### **Meta\_Expanded\_Data.Constructor:**

```

Sub Constructor()
    bin = new MemoryBlock(0)
End Sub

```

### **Meta\_Expanded\_Data.copy:**

```

Function copy() As Meta_Expanded_Data
    dim temp_raw as new Meta_Expanded_Data

    temp_raw.bin = new MemoryBlock(bin.size)
    for i as integer = 0 to bin.size - 1
        temp_raw.bin.byte(i) = bin.byte(i)
    next
    temp_raw.size = size
    redim temp_raw.reflexives(-1)
    for i as integer = 0 to UBound(reflexives)
        temp_raw.reflexives.Append Reflexives(i)
    next
    redim temp_raw.dependencies(-1)
    for i as integer = 0 to UBound(dependencies)
        temp_raw.dependencies.Append dependencies(i)
    next
    redim temp_raw.loneIDs(-1)
    for i as integer = 0 to UBound(loneIDs)
        temp_raw.loneIDs.Append loneIDs(i)
    next
    return temp_raw
End Function

```

### **Meta\_Expanded\_Data.Destructor:**

```

Sub Destructor()
    bin.Size = 0
End Sub
bin As memoryBlock

```

dependencies(-1) As tag\_reference

loneIDs(-1) As tag\_reference

reflexives(-1) As reflexive

size As Integer

End Class

Class Meta\_Raw

### **Meta\_Raw.Constructor:**

```
Sub Constructor()  
    bin = new MemoryBlock(0)  
End Sub
```

### **Meta\_Raw.copy:**

```
Function copy() As meta_raw  
    dim temp_raw as new meta_raw  
  
    temp_raw.bin = new MemoryBlock(bin.size)  
    for i as integer = 0 to bin.size - 1  
        temp_raw.bin.byte(i) = bin.byte(i)  
    next  
    temp_raw.size = size  
    temp_raw.str_info = str_info  
    temp_raw.offset = offset  
    return temp_raw  
End Function
```

### **Meta\_Raw.Destructor:**

```
Sub Destructor()  
    bin.Size = 0  
End Sub  
bin As memoryBlock
```

### **Meta\_Raw.offset:**

```
offset As Integer  
//only used for rebuilding  
//not important for meta editing  
size As Integer
```

str\_info As string

End Class

Class reflexive

### **reflexive.copy:**

```
Function copy() As reflexive  
    return self
```

```
End Function  
count As Integer
```

```
offset As Integer
```

```
translation As Integer
```

```
End Class  
Class tag_reference
```

#### **tag\_reference.copy:**

```
Function copy() As tag_Reference  
    return self  
End Function  
offset As Integer
```

```
tag_class As string
```

```
tag_name As string
```

```
End Class  
Class Meta_Pack
```

```
class1 As string
```

```
metas() As meta
```

```
name As string
```

```
End Class  
Class vFolder
```

#### **vFolder.Constructor:**

```
Sub Constructor()  
    redim child(-1)  
    redim item(-1)  
End Sub  
child(-1) As vFolder
```

```
item(-1) As variant
```

```
name As string
```

#### **vFolder Note: virtual folder**

virtual folder

basically, a simple way of organizing data into folders for listbox access

```
End Class  
Class preferences
```

### **preferences.Constructor:**

```
Sub Constructor()  
    byName = true  
    Ent_over_XML = true  
    explicit_bitmap = true  
    explicit_bitmap_address = ""  
    redim favored_plugins(-1)  
    scroll_edit = true  
    show_ent_hidden = false  
    Model_bitmaps = false  
    BSP_bitmaps = false  
    BSP_models = false  
    edit_by_offset = true
```

```
End Sub
```

### **preferences.read:**

```
Sub read(f as folderItem)  
    dim xdoc as new XmlDocument  
    xdoc.LoadXml f  
  
    dim node as XmlNode  
    for i as integer = 0 to xdoc.DocumentElement.ChildCount - 1  
        node = xdoc.DocumentElement.Child(i)  
        select case node.GetAttribute("name")  
            case "byName"  
                if node.GetAttribute("value") = "true" then  
                    byName = true  
                else  
                    byName = false  
                end  
  
            case "Ent_over_XML"  
                if node.GetAttribute("value") = "true" then  
                    Ent_over_XML = true  
                else  
                    Ent_over_XML = false  
                end  
  
            case "explicit_bitmap"  
                if node.GetAttribute("value") = "true" then  
                    explicit_bitmap = true  
                else  
                    explicit_bitmap = false  
                end  
  
            case "explicit_bitmap_address"  
                explicit_bitmap_address = node.GetAttribute("address")  
  
            case "scroll_edit"  
                if node.GetAttribute("value") = "true" then  
                    scroll_edit = true
```

```

        else
            scroll_edit = false
        end

    case "show_ent_hidden"
        if node.GetAttribute("value") = "true" then
            show_ent_hidden = true
        else
            show_ent_hidden = false
        end

    case "favored_plugins"
        favored_plugins = node.GetAttribute("folders").Split(":")

    case "BSP_models"
        if node.GetAttribute("value") = "true" then
            BSP_models = true
        else
            BSP_models = false
        end

    case "BSP_bitmaps"
        if node.GetAttribute("value") = "true" then
            BSP_bitmaps = true
        else
            BSP_bitmaps = false
        end

    case "Model_bitmaps"
        if node.GetAttribute("value") = "true" then
            Model_bitmaps = true
        else
            Model_bitmaps = false
        end

    case "edit_by_offset"
        if node.GetAttribute("value") = "true" then
            edit_by_offset = true
        else
            edit_by_offset = false
        end

    end select
next
End Sub

```

### **preferences.write:**

```

Sub write(f as folderItem)
    dim xdoc as XmlDocument
    xdoc = new XmlDocument

    dim item as XmlNode
    dim node as XmlNode

```

```

item = xdoc.AppendChild(xdoc.CreateElement("preference"))
item.SetAttribute("name", "Preferences")

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "byName")
if byName then
    node.SetAttribute("value", "true")
else
    node.SetAttribute("value", "false")
end

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "Ent_over_XML")
if Ent_over_XML then
    node.SetAttribute("value", "true")
else
    node.SetAttribute("value", "false")
end

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "explicit_bitmap")
if explicit_bitmap then
    node.SetAttribute("value", "true")
else
    node.SetAttribute("value", "false")
end

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "explicit_bitmap_address")
node.SetAttribute("address", explicit_bitmap_address)

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "scroll_edit")
if scroll_edit then
    node.SetAttribute("value", "true")
else
    node.SetAttribute("value", "false")
end

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "favored_plugins")
node.SetAttribute("folders", join(favored_plugins, ":"))

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "BSP_models")
if BSP_models then
    node.SetAttribute("value", "true")
else
    node.SetAttribute("value", "false")
end

node = item.AppendChild(xdoc.CreateElement("item"))
node.SetAttribute("name", "BSP_bitmaps")
if BSP_bitmaps then

```

```

        node.SetAttribute("value", "true")
    else
        node.SetAttribute("value", "false")
    end

    node = item.AppendChild(xdoc.CreateElement("item"))
    node.SetAttribute("name", "Model_bitmaps")
    if Model_bitmaps then
        node.SetAttribute("value", "true")
    else
        node.SetAttribute("value", "false")
    end

    node = item.AppendChild(xdoc.CreateElement("item"))
    node.SetAttribute("name", "show_ent_hidden")
    if show_ent_hidden then
        node.SetAttribute("value", "true")
    else
        node.SetAttribute("value", "false")
    end

    node = item.AppendChild(xdoc.CreateElement("item"))
    node.SetAttribute("name", "edit_by_offset")
    if edit_by_offset then
        node.SetAttribute("value", "true")
    else
        node.SetAttribute("value", "false")
    end

    xdoc.SaveXml(f)
End Sub
BSP_bitmaps As boolean

BSP_models As boolean

```

### **preferences.byName:**

```

byName As boolean = true
//controls whether to display meta by slicing name or by class

```

### **preferences.edit\_by\_offset:**

```

edit_by_offset As boolean = true
//controls whether it should display scrolling controls by offset

```

### **preferences.Ent\_over\_XML:**

```

Ent_over_XML As boolean
//if there are two plugins in the same folder one .ent one .xml, which one should be chosen

```

### **preferences.explicit\_bitmap:**

```

explicit_bitmap As boolean = false
//wether or not to use an explicit bitmap

```

### **preferences.explicit\_bitmap\_address:**

```
explicit_bitmap_address As string
//the file address of the explicit bitmap file
```

### **preferences.favored\_plugins():**

```
favored_plugins() As string
//an array of names of folders in order of preference for plugin access
Model_bitmaps As boolean
```

### **preferences.scroll\_edit:**

```
scroll_edit As boolean = true
//determines whether it uses the PC tools like scoller or the classic Eschaton menu system for meta editing
```

### **preferences.show\_ent\_hidden:**

```
show_ent_hidden As boolean = false
//show visible="false" data
End Class
Class map_expander
Inherits Thread
```

### **map\_expander.Run:**

```
Sub Run()
    dim w as new Map_Expand_Window
    w.tick(0)
    for i as integer = 0 to UBound(map.tags)
        map.expandTag(i)
        dim d as double = i
        w.tick( round((d*100)/UBound(map.tags)))
    next
    w.close

    map.expanded = true
End Sub
```

### **map\_expander.Constructor:**

```
Sub Constructor(byref in_map as h1.map)
    map = in_map
End Sub
map As h1.map

End Class
Class map_writer
Inherits Thread
```

### **map\_writer.Run:**

```
Sub Run()
    if garbage_collect then
        map.reorder_tags(true)
        if main.pref.byName then
            //close all windows
            for i as integer = 0 to main.ListBox1.ListCount-1
```



```

        if main.ListBox1.Expanded(i) then
            main.ListBox1.Expanded(i) = false
        end
    next
else
    main.update_list
end
end
dim w as new Map_Rebuild_Window
dim temp_bool as Boolean = false
temp_bool = map.write(f, w)
w.close

if temp_bool then
    MsgBox("Map Rebuilt Successfully")
else
    errorbox("There was an error rebuilding the map")
end
End Sub

```

### **map\_writer.init:**

```

Sub init(byref in_map as H1.map, in_f as folderItem, in_main as main_window, in_garbage_collect as boolean =
false)
    map = in_map
    f = in_f
    main = in_main
    garbage_collect = in_garbage_collect
End Sub
f As folderItem

garbage_collect As boolean

main As Main_Window

map As H1.map

End Class
Class CE_tag_replacement_thread
Inherits Thread

```

### **CE\_tag\_replacement\_thread.Run:**

```

Sub Run()
    //sanity check
    if not folder_f.Directory then
        finished = true
        return
    end

    if internalize then
        MsgBox("To internalize the data, please select a bitmaps.map file and a sounds.map file")
    end

    Dim bitmaps_f as FolderItem = nil

```

```

if internalize then
    dim title_str as string = "Select Bitmaps.map file"
    dim prompt_str as string = "Please select the bitmaps.map for data internalization"
    bitmaps_f= getOpenFolderitem_custom(title_str, prompt_str, folder_f, H1_Filetypes.HaloMapFile)
    if bitmaps_f = Nil then
        internalize = false
    end
end

Dim sounds_f as FolderItem = nil
if internalize then
    dim title_str as string = "Select Sounds.map file"
    dim prompt_str as string = "Please select the sounds.map for data internalization"
    sounds_f = getOpenFolderitem_custom(title_str, prompt_str, folder_f, H1_Filetypes.HaloMapFile)
    if sounds_f = Nil then
        internalize = false
    end
end

dim w as new Progress_Window("Replacing Tags")
w.show
w.tick("Tags remaining: " + str(tags_to_replace.ubound + 1), 0)

dim tags_remaining as integer = tags_to_replace.ubound + 1
dim total_tags as single = tags_remaining

for i as integer = 1 to folder_f.Count
    //quick check to make sure that not too much time is wasted loading a map file
    if tags_to_replace.Ubound < 0 then continue

    dim temp_f as FolderItem = folder_f.item(i)
    if temp_f.Type = H1_Filetypes.HaloMapFile.name then
        dim temp_map as new h1.map
        dim read_success as boolean = false
        read_success = temp_map.read(temp_f)
        if read_success then
            for j as integer = 0 to tags_to_replace.Ubound
                if temp_map.tagClassNameTable.haskey(tags_to_replace(j)) then
                    dim tag_index as integer = temp_map.tagClassNameTable.value(tags_to_replace(j))
                    temp_map.expandTag(tag_index)
                    if temp_map.tags(tag_index).expanded then
                        if internalize and temp_map.tags(tag_index).class1 = "mtib" then
                            dim internalize_success as boolean = temp_map.internalizeTag(tag_index, bitmaps_f)
                        end
                        if internalize and temp_map.tags(tag_index).class1 = "!dns" then
                            dim internalize_success as boolean = temp_map.internalizeTag(tag_index, sounds_f)
                        end
                        tags_to_replace(j) = "replaced"
                        replaced metas.Append temp_map.tags(tag_index)
                        tags_remaining = max(tags_remaining - 1, 0)
                        dim numerator as single = total_tags - tags_remaining
                        w.tick("Tags remaining: " + str(tags_remaining), (numerator/total_tags) * 100.0)
                    end
                end
            end
        end
    end
end

```

```

        next
        while tags_to_replace.IndexOf("replaced") > -1
            tags_to_replace.Remove(tags_to_replace.IndexOf("replaced"))
        wend
    end
end
next

w.close
finished = true
End Sub

```

### **CE\_tag\_replacement\_thread.init:**

```

Sub init(in_folder_f as folderItem, in_internalize as boolean, in_tags_to_replace() as string)
    internalize = in_internalize
    folder_f = in_folder_f
    tags_to_replace() = in_tags_to_replace()
End Sub
finished As boolean

Private folder_f As folderitem

Private internalize As boolean

replaced metas() As h1.meta

tags_to_replace() As string

End Class
End Module

```

## **Class Main\_Window**

Inherits Window

### **Main\_Window.Resized:**

```

Sub Resized()
    SmartSplitter1.behave
End Sub

```

### **Main\_Window.Resizing:**

```

Sub Resizing()
    SmartSplitter1.behave
End Sub

```

### **Main\_Window.CloseWindow:**

```

Function CloseWindow() As Boolean
    me.Close
    Return True

```

End Function

### **Main\_Window.ExpandMap:**

Function ExpandMap() As Boolean

```
dim row as integer = ListBox1.ListIndex
dim temp_str_split() as string = split(ListBox1.CellTag(row, 0), ":")
if temp_str_split.ubound <> 1 then Return True
dim type_split() as string = split(temp_str_split(UBound(temp_str_split)), "_")
dim index as integer = val(type_split(type_split.UBound))
if temp_str_split(0) = "maps_lite" then
    if index < 0 or index > maps_lite.Ubound then Return True
    dim temp_map as new H1.map
    if temp_map.read(maps_lite(index).fs) then
        if temp_map.expandMap then
            maps_lite.Remove(index)
            maps_expanded.Append temp_map
            self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False
            self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
            self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
            self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
            self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
            self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False
            EnableMenuItems
            update_list
        end
    end
end
```

Return True

End Function

### **Main\_Window.extractTag:**

Function extractTag() As Boolean

```
dim temp_str_split() as string = split(selected_path, ":")

select case temp_str_split(0)

case "extracted_meta"

case "maps_lite"
    dim temp_map as H1.Map = maps_lite(val(temp_str_split(1)))
    if not temp_map.tags(selected_index).expanded then
        temp_map.expandTag(selected_index)
    end
    if not temp_map.tags(selected_index).expanded then
        errorbox("Error: Could not extract tag")
        Return true
    end
    dim temp_meta_pack as new H1.Meta_Pack
    temp_meta_pack.metas.append temp_map.tags(selected_index)
    temp_meta_pack.class1 = temp_map.tags(selected_index).class1
    temp_meta_pack.name = temp_map.tags(selected_index).name
    extracted_meta_pack.append temp_meta_pack
    extracted_meta_folders.item.append UBound(extracted_meta_pack)
```

```

    update_list
    MsgBox("Tag Extracted Successfully")

case "maps_expanded"

    dim map_ind as integer = val(temp_str_split(1))
    if not maps_expanded(map_ind).tags(selected_index).expanded then
        errorbox("Error: Could not extract tag")
        Return true
    end
    dim temp_meta_pack as new H1.Meta_Pack
    temp_meta_pack.metas.append maps_expanded(map_ind).tags(selected_index)
    temp_meta_pack.class1 = maps_expanded(map_ind).tags(selected_index).class1
    temp_meta_pack.name = maps_expanded(map_ind).tags(selected_index).name
    extracted_meta_pack.append temp_meta_pack
    extracted_meta_folders.item.append UBound(extracted_meta_pack)
    update_list
    MsgBox("Tag Extracted Successfully")

case else

end select

return true
End Function

Main_Window.extractTag_external:
Function extractTag_external() As Boolean
    //needs to work out some info regarding local magic/version info
    //perhaps only if it's a bitmap, model or sound?
    //if so then create a new data file?
    //nah, when the tag is imported, check reflexives to determine the magic that was used

    dim temp_str_split() as string = split(selected_path, ":")

    Dim f as FolderItem
    f=SelectFolder
    if f = nil then return true

    select case temp_str_split(0)

    case "extracted_meta"

    case "maps_lite"
        dim temp_map as H1.Map = maps_lite(val(temp_str_split(1)))
        if not temp_map.tags(selected_index).expanded then
            temp_map.expandTag(selected_index)
        end
        if not temp_map.tags(selected_index).expanded then
            errorbox("Error: Could not extract tag")
            Return true
        end
        dim folder_f as FolderItem = generate_folders(f, temp_map.tags(selected_index).name)
        dim name_split() as string = temp_map.tags(selected_index).name.split("\")

```

```

dim meta_name as string = name_split(name_split.ubound)
dim meta_class as string = temp_map.tags(selected_index).class1.reverse
if folder_f = nil then
    errorbox("Error: Could not create folder")
    Return True
end
dim meta_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".meta")
dim xml_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".xml")

//write the meta data file
dim bw as BinaryStream = meta_f.CreateBinaryFile(H1_filetypes.EschatonMetaFile)
bw.LittleEndian = true
dim br as BinaryStream = new binarystream(temp_map.tags(selected_index).data.bin)
br.LittleEndian = true
br.Position = 0
while not br.EOF
    bw.WriteByte(br.ReadByte)
wend
br.close
bw.close

//write the xml file
dim xml_str as string = temp_map.tags(selected_index).generate_xml
dim xml_bw as BinaryStream = xml_f.CreateBinaryFile(H1_Filetypes.EschatonXMLFile)
xml_bw.LittleEndian = true
xml_bw.Position = 0
xml_bw.write(xml_str)
xml_bw.close

//write data files if
if temp_map.tags(selected_index).raw_data.ubound > -1 then
    dim data_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + " Data")
    data_f.CreateAsFolder
    for i as integer = 0 to UBound(temp_map.tags(selected_index).raw_data)
        dim raw_data_f as new FolderItem
        dim temp_str as string = temp_map.tags(selected_index).raw_data(i).str_info + ".bin"
        temp_str = temp_str.ReplaceAll(":", "|")
        raw_data_f = data_f.Child(temp_str)
        bw = raw_data_f.CreateBinaryFile(H1_filetypes.EschatonBINFile)
        bw.LittleEndian = true
        br = new BinaryStream(temp_map.tags(selected_index).raw_data(i).bin)
        br.LittleEndian = true
        br.Position = 0
        while not br.EOF
            bw.WriteByte(br.ReadByte)
        wend
        br.close
        bw.close
    next
end

MsgBox("Tag Extracted Successfully")

```

```

case "maps_expanded"

```

```

dim map_ind as integer = val(temp_str_split(1))
if not maps_expanded(map_ind).tags(selected_index).expanded then
    errorbox("Error: Could not extract tag")
    Return true
end
dim folder_f as FolderItem = generate_folders(f, maps_expanded(map_ind).tags(selected_index).name)
dim name_split() as string = maps_expanded(map_ind).tags(selected_index).name.split("\")
dim meta_name as string = name_split(name_split.ubound)
dim meta_class as string = maps_expanded(map_ind).tags(selected_index).class1.reverse
if folder_f = nil then
    errorbox("Error: Could not create folder")
    Return True
end
dim meta_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".meta")
dim xml_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".xml")

//write the meta data file
dim bw as BinaryStream = meta_f.CreateBinaryFile(H1_filetypes.EschatonMetaFile)
bw.LittleEndian = true
dim br as BinaryStream = new binarystream(maps_expanded(map_ind).tags(selected_index).data.bin)
br.LittleEndian = true
br.Position = 0
while not br.EOF
    bw.WriteByte(br.ReadByte)
wend
br.close
bw.close

//write the xml file
dim xml_str as string = maps_expanded(map_ind).tags(selected_index).generate_xml
dim xml_bw as BinaryStream = xml_f.CreateBinaryFile(H1_Filetypes.EschatonXMLFile)
xml_bw.LittleEndian = true
xml_bw.Position = 0
xml_bw.write(xml_str)
xml_bw.close

//write data files if
if maps_expanded(map_ind).tags(selected_index).raw_data.ubound > -1 then
    dim data_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + " Data")
    data_f.CreateAsFolder
    for i as integer = 0 to UBound(maps_expanded(map_ind).tags(selected_index).raw_data)
        dim raw_data_f as new FolderItem
        dim temp_str as string = maps_expanded(map_ind).tags(selected_index).raw_data(i).str_info + ".bin"
        temp_str = temp_str.ReplaceAll(":", "|")
        raw_data_f = data_f.Child(temp_str)
        bw = raw_data_f.CreateBinaryFile(H1_filetypes.EschatonBINFile)
        bw.LittleEndian = true
        br = new BinaryStream(maps_expanded(map_ind).tags(selected_index).raw_data(i).bin)
        br.LittleEndian = true
        br.Position = 0
        while not br.EOF
            bw.WriteByte(br.ReadByte)
        wend
    next i
end if

```

```

        br.close
        bw.close
    next
end

```

```

    MsgBox("Tag Extracted Successfully")

```

```

case else

```

```

end select

```

```

Return True

```

```

End Function

```

## **Main\_Window.FileCloseMap:**

```

Function FileCloseMap() As Boolean

```

```

    dim row as integer = Listbox1.ListIndex

```

```

    dim temp_str_split() as string = split(ListBox1.CellTag(row, 0), ":")

```

```

    if temp_str_split.ubound <> 1 then Return True

```

```

    dim type_split() as string = split(temp_str_split(UBound(temp_str_split)), "_")

```

```

    dim index as integer = val(type_split(type_split.UBound))

```

```

    select case temp_str_split(0)

```

```

    case "maps_expanded"

```

```

        if index < 0 or index > maps_expanded.Ubound then Return True

```

```

        maps_expanded.Remove(index)

```

```

        self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False

```

```

        update_list

```

```

    case "maps_lite"

```

```

        if index < 0 or index > maps_lite.Ubound then Return True

```

```

        maps_lite.Remove(index)

```

```

        self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False

```

```

        EnableMenuItems

```

```

        update_list

```

```

    end select

```

```

    if maps_lite.Ubound < 0 and maps_expanded.Ubound < 0 and extracted_meta_pack.Ubound < 0 then

```

```

        self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("TagSearch").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False

```

```

        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False

```

```

        EnableMenuItems

```



```
end
Return True
```

End Function

### **Main\_Window.FileLoadTag:**

```
Function FileLoadTag() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.EschatonMetaFile)
    if f = nil then return true

    load_tag(f)

    Return True
```

End Function

### **Main\_Window.FileLoadTagsRecursively:**

```
Function FileLoadTagsRecursively() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.EschatonMetaFile)
    if f = nil then return true
    dim folder_f as FolderItem = selectFolder_custom("Top Level Folder",_
    "Select the top level folder for the extracted meta data")
    if folder_f = nil then return true
    if f.urlpath.instr(folder_f.urlpath) = 0 then
        errorbox("Error: Selected meta is not within selected folder")
        return true
    end

    load_tags_recursively(f, folder_f)

    Return True
End Function
```

### **Main\_Window.Open:**

```
Function Open() As Boolean
    dim f as FolderItem = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
    if f = nil then Return true

    //windows issues
    '#if TargetWin32
    'for i as integer = 0 to UBound(maps_lite)
    'if f.absolutePath = maps_lite(i).fs.absolutePath then
    'errorbox("Error: map is already in use")
    'return true
    'end
    'next
    'for i as integer = 0 to UBound(maps_expanded)
    'if f.absolutePath = maps_expanded(i).fs.absolutePath then
    'errorbox("Error: map is already in use")
    'return true
    'end
    'next
    '#endif
```

Open(f)

Return True

End Function

### **Main\_Window.OpenLite:**

Function OpenLite() As Boolean

dim f as FolderItem = GetOpenFolderItem(H1\_Filetypes.HaloMapFile)

if f = nil then Return true

//windows issues

'#if TargetWin32

'for i as integer = 0 to UBound(maps\_lite)

'if f.absolutepath = maps\_lite(i).fs.absolutepath then

'errorbox("Error: map is already in use")

'return true

'end

'next

'for i as integer = 0 to UBound(maps\_expanded)

'if f.absolutepath = maps\_expanded(i).fs.absolutepath then

'errorbox("Error: map is already in use")

'return true

'end

'next

'#endif

OpenLite(f)

return true

End Function

### **Main\_Window.RecursiveextractTag:**

Function RecursiveextractTag() As Boolean

dim temp\_str\_split() as string = split(selected\_path, ":")

select case temp\_str\_split(0)

case "extracted\_meta"

case "maps\_lite"

dim temp\_map as H1.Map = maps\_lite(val(temp\_str\_split(1)))

dim tags\_to\_extract() as integer = array(selected\_index)

tags\_to\_extract = temp\_map.recursive\_tags(selected\_index, tags\_to\_extract)

dim temp\_meta\_pack as new H1.Meta\_Pack

temp\_meta\_pack.class1 = temp\_map.tags(selected\_index).class1

temp\_meta\_pack.name = temp\_map.tags(selected\_index).name

dim t as new recursive\_extract\_lite\_thread

t.init(self, temp\_map, temp\_meta\_pack, tags\_to\_extract)

t.run

case "maps\_expanded"

case else

```
end select
```

```
return true  
End Function
```

### **Main\_Window.RecursiveextractTag\_external:**

```
Function RecursiveextractTag_external() As Boolean  
    dim temp_str_split() as string = split(selected_path, ":")  
  
    Dim f as FolderItem  
    f=SelectFolder  
    if f = nil then return true  
  
    select case temp_str_split(0)  
  
        case "extracted_meta"  
  
        case "maps_lite"  
            dim temp_map as H1.Map = maps_lite(val(temp_str_split(1)))  
            dim tags_to_extract() as integer = array(selected_index)  
            tags_to_extract = temp_map.recursive_tags(selected_index, tags_to_extract)  
            dim t as new recursive_extract_lite_extern_thread  
            t.init(self, temp_map, tags_to_extract,f)  
            t.run  
  
        case "maps_expanded"  
  
        case else  
  
    end select  
  
    return true  
End Function
```

### **Main\_Window.TagSearch:**

```
Function TagSearch() As Boolean  
    dim w as new Tag_Search_Window  
  
    Return True  
  
End Function
```

### **Main\_Window.Constructor:**

```
Sub Constructor()  
    // Calling the overridden superclass constructor.  
    Super.Window  
  
    selected_path = "null"  
    selected_index = -1  
    pref = new H1.preferences  
    dim f_pref as FolderItem = GetFolderItem("").Child("Data")  
    if not f_pref.Exists then  
        f_pref.CreateAsFolder()  
    end if  
end Sub
```

```

end
f_pref = f_pref.Child("preferences.xml")
if f_pref.Exists then
    pref.read(f_pref)
end
extracted_meta_folders = new H1.vFolder
extracted_class_name = new Dictionary
//redim extracted_meta(-1)
redim extracted_meta_pack(-1)
redim maps_expanded(-1)
redim maps_lite(-1)

Listbox1.AddFolder("Maps")
Listbox1.AddFolder("Maps (Read/Write only)")
Listbox1.AddFolder("Extracted Tags")
Listbox1.CellTag(0,0) = "maps_expanded"
Listbox1.CellTag(1,0) = "maps_lite"
Listbox1.CellTag(2,0) = "extracted_meta"
Listbox1.RowPicture(0) = Folder
Listbox1.RowPicture(1) = Folder
Listbox1.RowPicture(2) = Folder
End Sub

```

### **Main\_Window.load\_tag:**

```

Sub load_tag(f as folderItem)
    if f = nil then return
    dim tag_name as string = f.name.ReplaceAll(".meta","")
    dim xml_f as FolderItem = f.Parent.Child(tag_name + ".xml")
    if not xml_f.Exists then
        errorbox("Error: Could not find XML file")
        return
    end
    dim data_flag as Boolean = false
    dim data_f as FolderItem = f.Parent.Child(tag_name + " Data")
    if data_f.Exists and data_f.Directory then data_flag = true

    dim temp_meta as h1.meta
    if data_flag then
        temp_meta = h1.load_meta(f, xml_f, data_f)
    else
        temp_meta = h1.load_meta(f, xml_f)
    end

    dim temp_meta_pack as new H1.Meta_Pack
    temp_meta_pack.metas.append temp_meta
    temp_meta_pack.class1 = temp_meta.class1
    temp_meta_pack.name = temp_meta.name
    extracted_meta_pack.append temp_meta_pack
    extracted_meta_folders.item.append UBound(extracted_meta_pack)
    update_list
    MsgBox("Tag Loaded Successfully!")
End Sub

```

### **Main\_Window.load\_tags\_recursively:**

```

Sub load_tags_recursively(tag_f as folderItem, folder_f as folderItem)
    dim t as new recursive_load_tag_thread
    t.init(tag_f, folder_f, self)
    t.run
End Sub

```

### **Main\_Window.Open:**

```

Sub Open(f as folderItem)
    dim m as new H1.map
    if not m.read(f) then
        //break
        Return
    end
    if not m.expandMap then
        //break
        Return
    end

    self.MenuBar.Child("MapMenu").Child("TagSearch").AutoEnable = True
    EnableMenuItems
    maps_expanded.Append(m)
    update_list
End Sub

```

### **Main\_Window.OpenLite:**

```

Sub OpenLite(f as folderItem)
    dim m as new H1.map
    if not m.read(f) then Return
    self.MenuBar.Child("MapMenu").Child("TagSearch").AutoEnable = True
    EnableMenuItems
    maps_lite.Append(m)
    update_list

    Return
End Sub

```

### **Main\_Window.select\_tag:**

```

Function select_tag(map_index as string, class_str as string, name as string) As boolean
    'dim err as new InvalidParentException
    'raise err

    //class_str is human readable, not how it appears in files

    //determine the path the selected tag should have
    dim final_path as string = Map_Index //start by giving it the correct map file
    dim path() as string = split(final_path, ":")
    dim current_folder as new H1.vFolder
    //highest level folder
    select case path(0)

    case "maps_expanded"
        if pref.byName then
            current_folder = maps_expanded(val(path(1))).meta_name_folders

```

```

else
    current_folder = maps_expanded(val(path(1))).meta_class_folders
end

case "maps_lite"
    if pref.byName then
        current_folder = maps_lite(val(path(1))).meta_name_folders
    else
        current_folder = maps_lite(val(path(1))).meta_class_folders
    end

case "extracted_meta"
    current_folder = extracted_meta_folders

end select
if pref.byName then
    dim path_internal() as string = split(name, "\")
    for i as integer = 0 to UBound(path_internal) - 1
        for j as integer = 0 to UBound(current_folder.child)
            if current_folder.child(j).name = path_internal(i) then
                final_path = final_path + ":c_" + str(j)
                current_folder = current_folder.child(j)
                exit for j
            end
        next
    next
    for i as integer = 0 to UBound(current_folder.item)
        select case path(0)
            case "maps_expanded"
                if maps_expanded(val(path(1))).tags(current_folder.item(i)).class1 = reverse(class_str) then
                    final_path = final_path + ":i_" + current_folder.item(i)
                    exit for i
                end
            case "maps_lite"
                if maps_lite(val(path(1))).tags(current_folder.item(i)).class1 = reverse(class_str) then
                    final_path = final_path + ":i_" + current_folder.item(i)
                    exit for i
                end
            case "extracted_meta"
                'if extracted_meta(current_folder.item(i)).class1 = reverse(class_str) then
                'final_path = final_path + ":i_" + current_folder.item(i)
                'exit for i
                'end
                if extracted_meta_pack(current_folder.item(i)).class1 = reverse(class_str) then
                    final_path = final_path + ":i_" + current_folder.item(i)
                    exit for i
                end
        end select
    next
else
    for i as integer = 0 to UBound(current_folder.child)
        dim temp_str() as string = split(current_folder.child(i).name, ":")
        if temp_str(0) = class_str then
            final_path = final_path + ":c_" + str(i)

```

```

        current_folder = current_folder.child(i)
    exit for i
end
next
for i as integer = 0 to UBound(current_folder.item)
    select case path(0)
    case "maps_expanded"
        if maps_expanded(val(path(1))).tags(current_folder.item(i)).name = name then
            final_path = final_path + ":i_" + current_folder.item(i)
            exit for i
        end
    case "maps_lite"
        if maps_lite(val(path(1))).tags(current_folder.item(i)).name = name then
            final_path = final_path + ":i_" + current_folder.item(i)
            exit for i
        end
    case "extracted_meta"
        'if extracted_meta(current_folder.item(i)).name = name then
        'final_path = final_path + ":i_" + current_folder.item(i)
        'exit for i
        'end
        if extracted_meta_pack(current_folder.item(i)).name = name then
            final_path = final_path + ":i_" + current_folder.item(i)
            exit for i
        end
    end select
next
end

```

//now that we have the appropriate path it's time to split it for folders then select the appropriate file

```

dim path_pieces() as string = split(final_path, ":")
dim opened_folders() as string
dim current_path as string = path_pieces(0)
opened_folders.append current_path
for i as integer = 1 to UBound(path_pieces) - 1
    current_path = current_path + ":" + path_pieces(i)
    opened_folders.append current_path
next

```

```

for i as integer = 0 to ListBox1.ListCount - 1
    if UBound(opened_folders) > -1 then
        if opened_folders(0) = Listbox1.CellTag(i, 0) then
            Listbox1.Expanded(i) = true
            opened_folders.Remove(0)
        end
    else
        exit for i
    end
next

```

```

for i as integer = 0 to ListBox1.ListCount - 1
    if Listbox1.CellTag(i, 0) = final_path then
        Listbox1.ListIndex = i
        dim temp as boolean = tag_selected(i,0)
    end if
next

```

```

        exit for i
    end
next

return true
End Function

```

### **Main\_Window.tag\_selected:**

```

Function tag_selected(row as integer, column as integer) As boolean
    dim temp_str_split() as string = split(ListBox1.CellTag(row, column), ":")
    dim type_split() as string = split(temp_str_split(UBound(temp_str_split)), "_")
    dim temp as integer = temp_str_split.ubound
    select case temp
case 0
    //top level folder selected
    self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False

    while TabPanel1.PanelCount > 0
        TabPanel1.Remove(0)
    wend
    TabPanel1.Append("Eschaton")
case 1
    //a map or tag collection
    select case temp_str_split(0)
case "maps_expanded"
        self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = True
        self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
        self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
        self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
        self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False
        dim index as integer = val(type_split(type_split.Ubound))
        if NOT (index < 0 or index > maps_expanded.Ubound) then
            while TabPanel1.PanelCount > 0
                TabPanel1.Remove(0)
            wend

            TabPanel1.Append("Map Information")
            dim w as new map_rebuild_control(self, index)
            w.Width = TabPanel1.Width - 4
            w.Height = TabPanel1.Height - 54
            w.EmbedWithinPanel(TabPanel1, 0, 2, 34)
            w.update

            //deprecated code for a control that does not fit into the Eschaton design paradigm
            'TabPanel1.Append("Advanced Rebuilding")
            'dim w1 as new map_rebuild_full_control(self, index)
            'w1.width = TabPanel1.Width - 4
            'w1.height = TabPanel1.Height - 54

```



```

        'w1.embedwithinpanel(TabPanel1, 1, 2, 34)
        'w1.update
        self.Refresh
    end

case "maps_lite"
    self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False
    dim index as integer = val(type_split(type_split.Ubound))
    if NOT (index < 0 or index > maps_lite.Ubound) then
        while TabPanel1.PanelCount > 0
            TabPanel1.Remove(0)
        wend
        TabPanel1.Append("Map Information")
        dim w as new map_control_RW(self, index)
        w.EmbedWithinPanel(TabPanel1, 0, 2, 34)
        self.Refresh
    end

case "extracted_meta"
    self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = True
    dim index as integer = val(type_split(type_split.Ubound))
    if NOT (index < 0 or index > extracted_meta_pack.Ubound) then
        while TabPanel1.PanelCount > 0
            TabPanel1.Remove(0)
        wend
        TabPanel1.Append("Extracted Tag Information")
        dim w as new extracted_tag_pack_control(self, index)
        w.Width = TabPanel1.Width - 4
        w.Height = TabPanel1.Height - 54
        w.EmbedWithinPanel(TabPanel1, 0, 2, 34)
        w.update
        self.Refresh
    end

end select

case else
    self.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
    while TabPanel1.PanelCount > 0
        TabPanel1.Remove(0)
    wend
    TabPanel1.Append("Eschaton")
    EnableMenuItems
    if type_split(0) <> "i" then Return False

```

```

//ok so we know that it's an item
dim temp_meta as new H1.meta
select case temp_str_split(0)
case "maps_expanded"
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = True
    selected_path = "maps_expanded:" + temp_str_split(1)
case "maps_lite"
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = True
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = True
    selected_path = "maps_lite:" + temp_str_split(1)
case "extracted_meta"
    self.MenuBar.Child("MapMenu").Child("extractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("extractTag_external").AutoEnable = False
    self.MenuBar.Child("MapMenu").Child("RecursiveextractTag_external").AutoEnable = False
    selected_path = "extracted_meta"
end select

selected_index = val(type_split(1))
update_tabs
end select

```

EnableMenuItems  
End Function

### **Main\_Window.update\_list:**

```

Sub update_list()
    //determine what was most recently selected
    dim selected_row_info as string
    if Listbox1.ListIndex = -1 then
        selected_row_info = "non"
    else
        selected_row_info = ListBox1.CellTag(ListBox1.ListIndex, 0)
    end

    //determine what was opened
    dim opened_folders(-1) as string
    for i as integer = 0 to Listbox1.ListCount-1
        if Listbox1.Expanded(i) then
            opened_folders.Append ListBox1.CellTag(i, 0)
        end
    next

    //close all folders
    for i as integer = 0 to ListBox1.ListCount-1
        if ListBox1.Expanded(i) then
            ListBox1.Expanded(i) = false
        end
    next
end Sub

```

```

    end
next

//reopen all folders
for i as integer = 0 to ListBox1.ListCount - 1
    if UBound(opened_folders) > -1 then
        if opened_folders(0) = Listbox1.CellTag(i, 0) then
            Listbox1.Expanded(i) = true
            opened_folders.Remove(0)
        end
    else
        exit for i
    end
next

//reselect selected row
dim selected as boolean = false
if selected_row_info <> "non" then
    for i as integer = 0 to ListBox1.ListCount - 1
        if Listbox1.CellTag(i, 0) = selected_row_info then
            selected = true
            Listbox1.Selected(i) = true
            exit for i
        end
    next
end

if not selected then
    while TabPanel1.PanelCount > 0
        TabPanel1.Remove(0)
    wend
    TabPanel1.Append("Eschaton")
end
End Sub

```

### **Main\_Window.update\_preferences:**

```

Sub update_preferences()
    dim old_pref_byName as boolean = pref.byName
    dim old_pref_scroll_edit as boolean = pref.scroll_edit

    //read the new shizzile
    dim f_pref as FolderItem = GetFolderItem("").Child("Data")
    if not f_pref.Exists then
        f_pref.CreateAsFolder()
    end
    f_pref = f_pref.Child("preferences.xml")
    if f_pref.Exists then
        pref.read(f_pref)
    end

    //check to see if changes are significant enough to require the closing of all folders
    if old_pref_scroll_edit <> pref.scroll_edit OR _
        old_pref_byName <> pref.byName then
        //close all folders
    end if
end Sub

```

```

    for i as integer = 0 to ListBox1.ListCount-1
        if ListBox1.Expanded(i) then
            ListBox1.Expanded(i) = false
        end
    next
end

//remove all existing panels
while TabPanel1.PanelCount > 0
    TabPanel1.Remove(0)
wend
TabPanel1.Append("Eschaton")
End Sub

```

### **Main\_Window.update\_tabs:**

```

Sub update_tabs()
    dim temp_str_split() as string = split(selected_path, ":")

    //NOTE TO SELF! ADD WAY OF AUTORESIZING MAIN CONTROLS FOR EXPANDED WINDOWS!
    //think this is taken care of

    select case temp_str_split(0)

    case "extracted_meta"

        //remove all existing panels
        while TabPanel1.PanelCount > 0
            TabPanel1.Remove(0)
        wend
        self.Refresh

    case "maps_lite"

        //remove all existing panels
        while TabPanel1.PanelCount > 0
            TabPanel1.Remove(0)
        wend

        if maps_lite(val(temp_str_split(1))).tags(selected_index).CE_flag = 1 then
            //add a new meta editing panel
            TabPanel1.Append("Meta Editor")
            dim c1 as new no_meta_control
            c1.Width = TabPanel1.Width - 4
            c1.Height = TabPanel1.Height - 54
            c1.EmbedWithinPanel(TabPanel1, 0, 2, 34)
        else
            dim f as folderItem = maps_lite(val(temp_str_split(1))).fs
            dim in_offset as integer = maps_lite(val(temp_str_split(1))).tags(selected_index).offset
            dim in_magic as integer = maps_lite(val(temp_str_split(1))).magic
            dim in_index_offset as integer = maps_lite(val(temp_str_split(1))).index_header.ind_offset
            dim in_vertex_offset as integer = maps_lite(val(temp_str_split(1))).index_header.vert_offset
            dim class_type as string = maps_lite(val(temp_str_split(1))).tags(selected_index).class1.reverse
            if class_type = "sbsp" then
                for i as integer = 0 to UBound(maps_lite(val(temp_str_split(1))).bsp_data)

```

```

        if maps_lite(val(temp_str_split(1))).bsp_data(i).tagID = _
            maps_lite(val(temp_str_split(1))).tags(selected_index).tagID then
            in_offset = maps_lite(val(temp_str_split(1))).bsp_data(i).offset
            in_magic = maps_lite(val(temp_str_split(1))).bsp_data(i).magic - in_offset
            exit for i
        end
    next
end
dim names() as string = split(maps_lite(val(temp_str_split(1))).tags(selected_index).name, "\")
dim name as string
if UBound(names) > -1 then
    name = names(UBound(names))
else
    name = "unknown"
end
dim size as integer = maps_lite(val(temp_str_split(1))).tags(selected_index).metasize
dim tagID as uint32 = maps_lite(val(temp_str_split(1))).tags(selected_index).tagID

//tag specific editing stuff
select case class_type
case "bitm"
    //need a check first to see if the file is internalized
    dim check_bitm as new bitmap_class_RW(f, f, in_offset, in_magic)
    check_bitm.read
    dim external as boolean = false
    for i as integer = 0 to UBound(check_bitm.image_info)
        if BitAnd(check_bitm.image_info(i).flags, &h100) = &h100 then
            external = true
            exit for i
        end
    next

    dim f_bitmap as FolderItem
    if external then
        if pref.explicit_bitmap then
            f_bitmap = GetFolderItem(pref.explicit_bitmap_address)
        else
            if maps_lite(val(temp_str_split(1))).bitmaps_f <> nil AND _
                maps_lite(val(temp_str_split(1))).bitmaps_f.exists then
                f_bitmap = maps_lite(val(temp_str_split(1))).bitmaps_f
            else
                f_bitmap = f.parent.child("bitmaps.map")
            end
        end
    end
    if f_bitmap = nil OR not f_bitmap.exists then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Select"
        d.CancelButton.Visible= True //show the Cancel button
        d.Message="Error: No bitmaps.map file found"
        d.Explanation="If you don't select a bitmaps.map file, you will not be able to view bitmaps. "
        b=d.ShowModal //display the dialog
        Select Case b //determine which button was pressed.

```

```

        Case d.ActionButton
            f_bitmap = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
            maps_lite(val(temp_str_split(1))).bitmaps_f = f_bitmap
        End select
    end
else
    f_bitmap = f
end
if f_bitmap <> nil And f_bitmap.exists AND (not external OR f_bitmap.name = "bitmaps.map") then
    dim temp as new bitmap_class_RW(f_bitmap, f, in_offset, in_magic)
    dim c1 as new bitm_editor_control_RW
    TabPanel1.Append("Bitmaps")
    c1.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 2, 24)
    c1.Init(temp, name)
end
special_id = 0

case "snd!"
    //need a check first to see if the file is internalized
    dim check_sound as new snd_class_RW(f, f, in_offset, in_magic)
    check_sound.read
    dim external as boolean = false
    for i as integer = 0 to UBound(check_sound.pitch)
        for j as integer = 0 to UBound(check_sound.pitch(i).permutations)
            if check_sound.pitch(i).permutations(j).internal = false then
                external = true
                exit for i
            end
        next
    next
    next

dim f_sound as FolderItem
if external then
    if maps_lite(val(temp_str_split(1))).sounds_f <> nil AND _
        maps_lite(val(temp_str_split(1))).sounds_f.exists then
        f_sound = maps_lite(val(temp_str_split(1))).sounds_f
    else
        f_sound = f.parent.child("sounds.map")
    end
    if f_sound = nil OR not f_sound.exists then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Select"
        d.CancelButton.Visible= True //show the Cancel button
        d.Message="Error: No sounds.map file found"
        d.Explanation="If you don't select a sounds.map file, you will not be able to edit sounds. "
        b=d.ShowModal //display the dialog
        Select Case b //determine which button was pressed.
        Case d.ActionButton
            f_sound = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
            maps_lite(val(temp_str_split(1))).sounds_f = f_sound
        End select
    end
end

```

```

else
    f_sound = f
end
if f_sound <> nil And f_sound.exists AND (f_sound.name = "sounds.map" or not external) then
    dim temp as new snd_class_RW(f_sound, f, in_offset, in_magic)
    dim c1 as new snd_editor_control_RW
    TabPanel1.Append("Sounds")
    c1.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 2, 24)
    c1.Init(temp)
end
special_id = 1

case "mod2"
'dim check_model1 as new mod2_class(f, in_offset, in_magic)
'check_model1.read
'break
TabPanel1.Append("Model")

'dim in_fullname as string = name
'dim m_control as new Mod2_control
'm_control.init(f, in_magic, in_offset, in_index_offset, in_vertex_offset, class_type, in_fullname)
'm_control.embedwithinpanel(TabPanel1, TabPanel1.PanelCount - 1, 2, 24)

//need to figure out how to not have Rb3DSpace show up always
dim check_model as new model
check_model.read(f, in_magic, in_offset, in_index_offset, in_vertex_offset)
dim m_control as new Model_control
m_control.Width = TabPanel1.Width - 4
m_control.Height = TabPanel1.Height - 54
m_control.embedwithinpanel(TabPanel1, TabPanel1.PanelCount - 1, 2, 24)
m_control.init(check_model, f)
special_control = m_control
special_id = 2

case "scnr"
    TabPanel1.Append("Scenario Editor")

    dim scnr_control as new BSP_display
    scnr_control.Width = TabPanel1.Width - 4
    scnr_control.Height = TabPanel1.Height - 54
    scnr_control.embedwithinpanel(TabPanel1, TabPanel1.PanelCount - 1, 2, 24)
    scnr_control.init(f)
    special_control = scnr_control
    special_id = 3

case "ustr"
    TabPanel1.Append("Unicode String")

    dim u_control as new ustr_edit_control
    u_control.Width = TabPanel1.Width - 4
    u_control.Height = TabPanel1.Height - 54
    u_control.embedwithinpanel(TabPanel1, TabPanel1.PanelCount - 1, 2, 24)
    u_control.init(f, in_magic, in_offset)
    special_id = 4

```

```

case else
    special_id = -1

end select

//self.Refresh

dim f_top as folderItem = GetFolderItem("").Child("Plugins")
dim f_xml as FolderItem = nil
dim entity_style as boolean
//first sort through all the possible preferred plugin folders
for i as integer = 0 to UBound(pref.favored_plugins)
    dim temp_f as FolderItem = f_top.child(pref.favored_plugins(i))
    if temp_f.exists and temp_f.directory then
        dim f_xml_folder as FolderItem = temp_f
        //ugly if elses based on preferences
        if pref.Ent_over_XML then
            if f_xml_folder.child(class_type + ".ent").exists then
                f_xml = f_xml_folder.child(class_type + ".ent")
                entity_style = true
                exit for i
            else
                if f_xml_folder.child(class_type + ".xml").exists then
                    f_xml = f_xml_folder.child(class_type + ".xml")
                    entity_style = false
                    exit for i
                end
            end
        else
            if f_xml_folder.child(class_type + ".xml").exists then
                f_xml = f_xml_folder.child(class_type + ".xml")
                entity_style = false
                exit for i
            else
                if f_xml_folder.child(class_type + ".ent").exists then
                    f_xml = f_xml_folder.child(class_type + ".ent")
                    entity_style = true
                    exit for i
                end
            end
        end
    end
end
next
if f_xml = nil then
    for i as integer = 1 to f_top.count
        dim temp_f as FolderItem = f_top.item(i)
        if temp_f.exists and temp_f.directory then
            dim f_xml_folder as FolderItem = temp_f
            //ugly if elses based on preferences
            if pref.Ent_over_XML then
                if f_xml_folder.child(class_type + ".ent").exists then
                    f_xml = f_xml_folder.child(class_type + ".ent")
                    entity_style = true
                end
            end
        end
    end
end

```



```

        exit for i
    else
        if f_xml_folder.child(class_type + ".xml").exists then
            f_xml = f_xml_folder.child(class_type + ".xml")
            entity_style = false
            exit for i
        end
    end
else
    if f_xml_folder.child(class_type + ".xml").exists then
        f_xml = f_xml_folder.child(class_type + ".xml")
        entity_style = false
        exit for i
    else
        if f_xml_folder.child(class_type + ".ent").exists then
            f_xml = f_xml_folder.child(class_type + ".ent")
            entity_style = true
            exit for i
        end
    end
end
end
next
end
if f_xml = nil then
    dim c1 as new no_plugin_control
    c1.Width = TabPanel1.Width - 4
    c1.Height = TabPanel1.Height - 54
    TabPanel1.Append("Meta Editor")
    c1.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 2, 34)
else
    if pref.scroll_edit then
        //use a scrolling control
        //dim c1 as new Container_Scroller
        dim c1 as new main_control_RWS(f, in_offset, in_magic)
        c1.Width = TabPanel1.Width - 4
        c1.Height = TabPanel1.Height - 54
        TabPanel1.Append("Meta Editor")
        c1.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 2, 34)
        dim temp_map as H1.map = maps_lite(val(temp_str_split(1)))
        c1.init(f_xml, entity_style, pref.show_ent_hidden, pref.edit_by_offset, temp_map)
        c1.read
        //c1.calculated_height = m.Height + 14 + 20
        //c1.calculated_width = m.width + 20 + 20
        //c1.Update
    else
        dim c1 as new main_control_RWC(f, in_offset, in_magic)
        c1.Width = TabPanel1.Width - 4
        c1.Height = TabPanel1.Height - 54
        TabPanel1.Append("Meta Editor")
        c1.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 2, 34)
        dim temp_map as H1.map = maps_lite(val(temp_str_split(1)))
        c1.init(f_xml, entity_style, pref.show_ent_hidden, temp_map)
    end
end

```

```

end
//self.Refresh

//this section is for adding a dependency swapper
dim d_swap as new dependency_swapper_control_RW(f,maps_lite(val(temp_str_split(1))), _
in_offset, size, me, val(temp_str_split(1)), tagID)
TabPanel1.Append("Reference Swapper")
d_swap.Height = TabPanel1.Height - 54
d_swap.Width = TabPanel1.Width - 10
d_swap.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 10, 34)
d_swap.read

//self.Refresh
end

self.Refresh

case "maps_expanded"

//remove all existing panels
while TabPanel1.PanelCount > 0
    TabPanel1.Remove(0)
wend

dim temp_control as new expanded_tag_control(self, val(temp_str_split(1)), selected_index)
TabPanel1.Append("Tag Information")
temp_control.Height = TabPanel1.Height - 54
temp_control.Width = TabPanel1.Width - 10
temp_control.EmbedWithinPanel(TabPanel1, 0, 2, 34)
temp_control.update

dim d_swap as new dependency_swapper_control_EX(self, val(temp_str_split(1)), selected_index)
TabPanel1.Append("Reference Swapper")
d_swap.Height = TabPanel1.Height - 54
d_swap.Width = TabPanel1.Width - 10
d_swap.EmbedWithinPanel(TabPanel1, TabPanel1.panelcount - 1, 10, 34)
d_swap.update

self.Refresh

case else
    return

end select
End Sub
extracted_class_name As dictionary

extracted_meta_folders As H1.vFolder

extracted_meta_pack(-1) As H1.Meta_Pack

maps_expanded(-1) As H1.map

maps_lite(-1) As H1.map

```

pref As H1.preferences

selected\_index As integer

selected\_path As string

special\_control As containercontrol

special\_id As Integer

## **Main\_Window Note: Extracted meta editing**

Extracted meta editing

SEARCH FOR "extracted\_meta("

remove everything "extracted\_meta\_pack("

## **Main\_Window Control ListBox1:**

Sub ExpandRow(row As Integer)

dim path() as string = split(listbox1.CellTag(row, 0), ":")

select case UBound(path)

case -1

return

//not really necessary

case 0

//just expand the folder to show the contents of that array

select case path(0)

case "maps\_expanded"

for i as integer = 0 to UBound(maps\_expanded)

Listbox1.AddFolder(maps\_expanded(i).fs.displayname)

//Listbox1.AddRow(maps\_expanded(i).fs.displayname) //temporary for the purposes of rebuilds only, no expanded editing

Listbox1.CellTag(Listbox1.LastIndex, 0) = "maps\_expanded:" + str(i)

Listbox1.RowPicture(Listbox1.LastIndex) = folder

//Listbox1.RowPicture(ListBox1.LastIndex) = green

next

case "maps\_lite"

for i as integer = 0 to UBound(maps\_lite)

Listbox1.AddFolder(maps\_lite(i).fs.displayname)

Listbox1.CellTag(Listbox1.LastIndex, 0) = "maps\_lite:" + str(i)

Listbox1.RowPicture(ListBox1.LastIndex) = folder

next

case "extracted\_meta"

for i as integer = 0 to UBound(extracted\_meta\_folders.child)

Listbox1.AddFolder(extracted\_meta\_folders.child(i).name)

Listbox1.CellTag(Listbox1.LastIndex, 0) = "extracted\_meta:c\_" + str(i)

Listbox1.RowPicture(ListBox1.LastIndex) = folder

```

next
for i as integer = 0 to UBound(extracted_meta_folders.item)
    //dim temp_str as string = extracted_meta(extracted_meta_folders.item(i)).name
    dim temp_str as string
    if extracted_meta_pack(extracted_meta_folders.item(i)).metas.ubound > 0 then
        temp_str = reverse(extracted_meta_pack(extracted_meta_folders.item(i)).class1) _
        + ": " + extracted_meta_pack(extracted_meta_folders.item(i)).name + " (Recursive)"
    else
        temp_str = reverse(extracted_meta_pack(extracted_meta_folders.item(i)).class1) _
        + ": " + extracted_meta_pack(extracted_meta_folders.item(i)).name
    end
    Listbox1.AddRow(temp_str)
    Listbox1.CellTag(Listbox1.LastIndex, 0) = "extracted_meta:i_" + str(extracted_meta_folders.item(i))
    Listbox1.RowPicture(Listbox1.LastIndex) = black
next
end select

case else
    //find the data and show it!

    dim current_folder as new H1.vFolder
    select case path(0)

    case "maps_expanded"
        if pref.byName then
            current_folder = maps_expanded(val(path(1))).meta_name_folders
        else
            current_folder = maps_expanded(val(path(1))).meta_class_folders
        end

    case "maps_lite"
        if pref.byName then
            current_folder = maps_lite(val(path(1))).meta_name_folders
        else
            current_folder = maps_lite(val(path(1))).meta_class_folders
        end

    case "extracted_meta"
        current_folder = extracted_meta_folders

    end select

    for i as integer = 2 to UBound(path)
        dim split_str() as string = split(path(i), "_")
        if split_str(0) = "c" then
            //it's a child folder
            current_folder = current_folder.child(val(split_str(1)))
        else
            //it's an item
            //this shouldn't trigger
            break
        end
    next

```

```

//sort the folders
dim sort_ar() as string
dim sort_ar_ind() as integer
for i as integer = 0 to UBound(current_folder.child)
    sort_ar.append current_folder.child(i).name
    sort_ar_ind.append i
next
sort_ar.sortwith(sort_ar_ind)

for i as integer = 0 to UBound(sort_ar_ind)
    Listbox1.AddFolder(current_folder.child(sort_ar_ind(i)).name)
    Listbox1.CellTag(Listbox1.LastIndex, 0) = listbox1.CellTag(row, 0) + ":c_" + str(sort_ar_ind(i))
    Listbox1.RowPicture(ListBox1.LastIndex) = folder
next

//sort the items
redim sort_ar(-1)
redim sort_ar_ind(-1)
for i as integer = 0 to UBound(current_folder.item)
    dim temp_str as string
    select case path(0)
    case "maps_expanded"
        if pref.byName then
            dim temp_split_str() as string = split(maps_expanded(val(path(1))).tags(current_folder.item(i)).name, "\")
            temp_str = temp_split_str(UBound(temp_split_str)) + "." +
                reverse(maps_expanded(val(path(1))).tags(current_folder.item(i)).class1)
        else
            temp_str = maps_expanded(val(path(1))).tags(current_folder.item(i)).name
        end
    case "maps_lite"
        if pref.byName then
            dim temp_split_str() as string = split(maps_lite(val(path(1))).tags(current_folder.item(i)).name, "\")
            temp_str = temp_split_str(UBound(temp_split_str)) + "." +
                reverse(maps_lite(val(path(1))).tags(current_folder.item(i)).class1)
        else
            temp_str = maps_lite(val(path(1))).tags(current_folder.item(i)).name
        end
    case "extracted_meta"
        //temp_str = extracted_meta(current_folder.item(i)).name
        temp_str = extracted_meta_pack(current_folder.item(i)).name
    end select
    sort_ar.append temp_str
    sort_ar_ind.append i
next
sort_ar.sortwith(sort_ar_ind)

for i as integer = 0 to UBound(sort_ar_ind)
    select case path(0)

    case "maps_expanded"
        dim temp_str as string
        if pref.byName then

```

```

        dim temp_split_str() as string =
        split(maps_expanded(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).name, "\")
        temp_str = temp_split_str(UBound(temp_split_str)) + "." +
        reverse(maps_expanded(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).class1)
    else
        temp_str = maps_expanded(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).name
    end
    Listbox1.AddRow(temp_str)
    Listbox1.CellTag(Listbox1.LastIndex, 0) = listbox1.CellTag(row, 0) + ":i_" +
    str(current_folder.item(sort_ar_ind(i)))
    Listbox1.RowPicture(Listbox1.LastIndex) = green
case "maps_lite"
    dim temp_str as string
    if pref.byName then
        dim temp_split_str() as string =
        split(maps_lite(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).name, "\")
        temp_str = temp_split_str(UBound(temp_split_str)) + "." +
        reverse(maps_lite(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).class1)
    else
        temp_str = maps_lite(val(path(1))).tags(current_folder.item(sort_ar_ind(i))).name
    end
    Listbox1.AddRow(temp_str)
    Listbox1.CellTag(Listbox1.LastIndex, 0) = listbox1.CellTag(row, 0) + ":i_" +
    str(current_folder.item(sort_ar_ind(i)))
    Listbox1.RowPicture(Listbox1.LastIndex) = red
case "extracted_meta"
    //dim temp_str as string = extracted_meta(current_folder.item(sort_ar_ind(i))).name
    dim temp_str as string = extracted_meta_pack(current_folder.item(sort_ar_ind(i))).name
    Listbox1.AddRow(temp_str)
    Listbox1.CellTag(Listbox1.LastIndex, 0) = listbox1.CellTag(row, 0) + ":i_" +
    str(current_folder.item(sort_ar_ind(i)))
    Listbox1.RowPicture(Listbox1.LastIndex) = black
end select
next

end select
End Sub

Function KeyDown(Key As String) As Boolean
    //31 for down 30 for up
    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end
end
end

```

```

if ok then
    return tag_selected(row, column)
end
End Function

```

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    return tag_selected(row, column)
End Function

```

## Main\_Window Control TabPanel1:

```

Sub Change()
    //only for dealing with special tags that have 3Dspaces which cause OSX graphical issues

    select case me.Value

case 0 //special tag maybe
    dim temp_str_split() as string = split(selected_path, ":")
    select case temp_str_split(0)
case "maps_lite"
        dim f as folderItem = maps_lite(val(temp_str_split(1))).fs
        dim in_offset as integer = maps_lite(val(temp_str_split(1))).tags(selected_index).offset
        dim in_magic as integer = maps_lite(val(temp_str_split(1))).magic
        dim in_index_offset as integer = maps_lite(val(temp_str_split(1))).index_header.ind_offset
        dim in_vertex_offset as integer = maps_lite(val(temp_str_split(1))).index_header.vert_offset
        dim class_type as string = maps_lite(val(temp_str_split(1))).tags(selected_index).class1.reverse
        dim names() as string = split(maps_lite(val(temp_str_split(1))).tags(selected_index).name, "\")
        dim name as string
        if UBound(names) > -1 then
            name = names(UBound(names))
        else
            name = "unknown"
        end
        dim size as integer = maps_lite(val(temp_str_split(1))).tags(selected_index).metasize
        dim tagID as uint32 = maps_lite(val(temp_str_split(1))).tags(selected_index).tagID

        if maps_lite(val(temp_str_split(1))).tags(selected_index).CE_flag = 1 then return
        select case special_id
case 0
            //bitm
case 1
            //snd!
case 2
            //mod2
            dim check_model as new model
            check_model.read(f, in_magic, in_offset, in_index_offset, in_vertex_offset)
            dim m_control as new Model_control
            m_control.Width = TabPanel1.Width - 4
            m_control.Height = TabPanel1.Height - 54
            m_control.embedwithinpanel(TabPanel1, 0, 2, 24)
            m_control.init(check_model, f)
            special_id = 2
            special_control = m_control
case 3

```

```

        //scnr
        dim scnr_control as new BSP_display
        scnr_control.Width = TabPanel1.Width -4
        scnr_control.Height = TabPanel1.Height - 54
        scnr_control.embedwithinpanel(TabPanel1, 0, 2, 24)
        scnr_control.init(f)
        special_control = scnr_control
        special_id = 3
    end select
end select

case else
    if special_control <> nil and special_control isa model_control then
        try
            special_control.Close
        catch NilObjectException
        end try
    end
    if special_control <> nil and special_control isa BSP_display then
        try
            special_control.Close
        catch NilObjectException
        end try
    end
end
End Sub
End Class

```

## **Class recursive\_extract\_lite\_thread**

Inherits Thread

### **recursive\_extract\_lite\_thread.Run:**

```

Sub Run()
    //do a first pass of the meta files to see if any of them are CE indexed tags
    dim CE_flagged(-1) as integer
    for i as integer = 0 to UBound(tags_to_extract)
        if temp_map.tags(tags_to_extract(i)).CE_flag = 1 then
            CE_flagged.append i
        end
    next

    dim temp_metas(-1) as h1.meta

    if CE_flagged.ubound > -1 then
        Dim d as New MessageDialog //declare the MessageDialog object
        Dim b as MessageDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Continue"
        d.CancelButton.Visible= True //show the Cancel button
        d.AlternateActionButton.Visible= True //show the "Replace" button
        d.AlternateActionButton.Caption="Replace"
        d.Message="Warning: Indexed tags cannot be extracted"
    end if
End Sub

```



```

d.Explanation="Indexed tags do not exist within map files." + _
" Select continue to extract without them. Select replace to use tags that do exist from other map files."
b=d.ShowModal    //display the dialog
Select Case b //determine which button was pressed.
Case d.ActionButton
    //user pressed Continue, do nothing
Case d.AlternateActionButton
    //user pressed Replace, open the replacement window
    dim tags_to_replace(-1) as string
    for i as integer = 0 to UBound(CE_flagged)
        dim class_str as string = temp_map.tags(tags_to_extract(CE_flagged(i))).class1
        dim name_str as string = temp_map.tags(tags_to_extract(CE_flagged(i))).name
        tags_to_replace.append class_str + ":" + name_str
    next
    dim replaced metas() as h1.meta = CE_replace(tags_to_replace)
    for i as integer = 0 to replaced_metas.ubound
        temp_metas.append replaced_metas(i)
    next

Case d.CancelButton
    //user pressed Cancel, kill the process
    return
End select
end

dim w as new Progress_Window("Tag Extraction")
w.tick("Recursively Extracting Tags", 0)
for i as integer = 0 to UBound(tags_to_extract)
    //this will only happen if a CE tag was found and the user selected continue
    if CE_flagged.indexOf(i) <> -1 then continue

    dim status_num as double = i+1
    dim status_denom as double = UBound(tags_to_extract)+1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Recursively Extracting Tags", status)
    if not temp_map.tags(tags_to_extract(i)).expanded then
        temp_map.expandTag(tags_to_extract(i))
    end
    if not temp_map.tags(tags_to_extract(i)).expanded then
        w.close
        errorbox("Error: Could not extract " + reverse(temp_map.tags(tags_to_extract(i)).class1) + _
            ": " + temp_map.tags(tags_to_extract(i)).name)
        Return
    end
    temp_meta_pack.metas.append temp_map.tags(tags_to_extract(i))
next

//this only works because the first tag should not be a non-existent tag
for i as integer = 0 to UBound(temp_metas)
    temp_meta_pack.metas.append temp_metas(i)
next

main.extracted_meta_pack.append temp_meta_pack

```

```

    main.extracted_meta_folders.item.append UBound(main.extracted_meta_pack)
    main.update_list
    w.close
    MsgBox("Tags Extracted Successfully")
End Sub

```

### **recursive\_extract\_lite\_thread.init:**

```

Sub init(byref in_main as main_window, byref in_temp_map as h1.map, byref in_temp_meta_pack as H1.meta_pack,
in_tags_to_extract() as integer)
    main = in_main
    temp_map = in_temp_map
    temp_meta_pack = in_temp_meta_pack
    tags_to_extract = in_tags_to_extract
End Sub
main As Main_Window

```

```
tags_to_extract() As Integer
```

```
temp_map As h1.map
```

```
temp_meta_pack As H1.meta_pack
```

```
End Class
```

## **Class recursive\_extract\_lite\_extern\_thread**

Inherits Thread

### **recursive\_extract\_lite\_extern\_thread.Run:**

```

Sub Run()
    //do a first pass of the meta files to see if any of them are CE indexed tags
    dim CE_flagged(-1) as integer
    for i as integer = 0 to UBound(tags_to_extract)
        if temp_map.tags(tags_to_extract(i)).CE_flag = 1 then
            CE_flagged.append i
        end
    next

    dim temp_metas(-1) as h1.meta
    if CE_flagged.ubound > -1 then
        Dim d as New MatDialog //declare the MatDialog object
        Dim b as MatDialogButton //for handling the result
        d.icon=MessageDialog.GraphicCaution //display warning icon
        d.ActionButton.Caption="Continue"
        d.CancelButton.Visible= True //show the Cancel button
        d.AlternateActionButton.Visible= True //show the "Replace" button
        d.AlternateActionButton.Caption="Replace"
        d.Message="Warning: Indexed tags cannot be extracted"
        d.Explanation="Indexed tags do not exist within map files." + _
        " Select continue to extract without them. Select replace to use tags that do exist from other map files."
        b=d.ShowModal //display the dialog
        Select Case b //determine which button was pressed.
        Case d.ActionButton

```

```

    //user pressed Continue, do nothing
Case d.AlternateActionButton
    //user pressed Replace, open the replacement window
    dim tags_to_replace(-1) as string
    for i as integer = 0 to UBound(CE_flagged)
        dim class_str as string = temp_map.tags(tags_to_extract(CE_flagged(i))).class1
        dim name_str as string = temp_map.tags(tags_to_extract(CE_flagged(i))).name
        tags_to_replace.append class_str + ":" + name_str
    next
    dim replaced metas() as h1.meta = CE_replace(tags_to_replace)
    for i as integer = 0 to replaced_metas.ubound
        temp_metas.append replaced_metas(i)
    next
Case d.CancelButton
    //user pressed Cancel, kill the process
    return
End select
end

dim w as new Progress_Window("Tag Extraction")
w.tick("Recursively Extracting Tags", 0)

dim bw as BinaryStream
dim br as BinaryStream

for i as integer = 0 to UBound(tags_to_extract)
    //this will only happen if a CE tag was found and the user selected continue
    if CE_flagged.indexOf(i) <> -1 then continue

    dim status_num as double = i+1
    dim status_denom as double = UBound(tags_to_extract)+1
    dim status_dec as double = status_num/status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Recursively Extracting Tags", status)
    if not temp_map.tags(tags_to_extract(i)).expanded then
        temp_map.expandTag(tags_to_extract(i))
    end
    if not temp_map.tags(tags_to_extract(i)).expanded then
        w.close
        errorbox("Error: Could not extract " + reverse(temp_map.tags(tags_to_extract(i)).class1) + _
            ": " + temp_map.tags(tags_to_extract(i)).name)
        Return
    end
    temp_metas.append temp_map.tags(tags_to_extract(i))
next

for i as integer = 0 to UBound(temp_metas)
    dim folder_f as FolderItem = generate_folders(f, temp_map.tags(tags_to_extract(i)).name)
    dim name_split() as string = temp_map.tags(tags_to_extract(i)).name.split("\")
    dim meta_name as string = name_split(name_split.ubound)
    dim meta_class as string = temp_map.tags(tags_to_extract(i)).class1.reverse
    if folder_f = nil then
        errorbox("Error: Could not create folder")
        Return
    end

```

```

end
dim meta_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".meta")
dim xml_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + ".xml")

//write the meta data file
bw = meta_f.CreateBinaryFile(H1_filetypes.EschatonMetaFile)
bw.LittleEndian = true
br = new binarystream(temp_map.tags(tags_to_extract(i)).data.bin)
br.LittleEndian = true
br.Position = 0
while not br.EOF
    bw.WriteByte(br.ReadByte)
wend
br.close
bw.close

//write the xml file
dim xml_str as string = temp_map.tags(tags_to_extract(i)).generate_xml
dim xml_bw as BinaryStream = xml_f.CreateBinaryFile(H1_Filetypes.EschatonXMLFile)
xml_bw.LittleEndian = true
xml_bw.Position = 0
xml_bw.write(xml_str)
xml_bw.close

//write data files if
if temp_map.tags(tags_to_extract(i)).raw_data.ubound > -1 then
    dim data_f as FolderItem = folder_f.Child(meta_name + "." + meta_class + " Data")
    data_f.CreateAsFolder
    for j as integer = 0 to UBound(temp_map.tags(tags_to_extract(i)).raw_data)
        dim raw_data_f as new FolderItem
        dim temp_str as string = temp_map.tags(tags_to_extract(i)).raw_data(j).str_info + ".bin"
        temp_str = temp_str.ReplaceAll(":", "|")
        raw_data_f = data_f.Child(temp_str)
        bw = raw_data_f.CreateBinaryFile(H1_filetypes.EschatonBINFile)
        bw.LittleEndian = true
        br = new BinaryStream(temp_map.tags(tags_to_extract(i)).raw_data(j).bin)
        br.LittleEndian = true
        br.Position = 0
        while not br.EOF
            bw.WriteByte(br.ReadByte)
        wend
        br.close
        bw.close
    next
end
next

w.close
MsgBox("Tags Extracted Successfully")
End Sub

```

### **recursive\_extract\_lite\_extern\_thread.init:**

```

Sub init(byref in_main as main_window, byref in_temp_map as h1.map, in_tags_to_extract() as integer, in_f as
folderItem)

```

```

    main = in_main
    temp_map = in_temp_map
    tags_to_extract = in_tags_to_extract
    f = in_f
End Sub
f As folderItem

```

```

main As Main_Window

```

```

tags_to_extract() As Integer

```

```

temp_map As h1.map

```

```

End Class

```

## **Class recursive\_load\_tag\_thread**

Inherits Thread

### **recursive\_load\_tag\_thread.Run:**

```

Sub Run()
    dim metas(-1) as h1.meta
    dim meta_queue(-1) as string
    dim processed_metas(-1) as string

    //process the very first meta
    dim tag_name as string = meta_f.name.ReplaceAll(".meta","")
    dim xml_f as FolderItem = meta_f.Parent.Child(tag_name + ".xml")
    if not xml_f.Exists then
        errorbox("Error: Could not find XML file")
        return
    end
    dim data_flag as Boolean = false
    dim data_f as FolderItem = meta_f.Parent.Child(tag_name + " Data")
    if data_f.Exists and data_f.Directory then data_flag = true

    dim start_meta as h1.meta
    if data_flag then
        start_meta = h1.load_meta(meta_f, xml_f, data_f)
    else
        start_meta = h1.load_meta(meta_f, xml_f)
    end

    dim w as new Waiting_Window("Loading Tags")
    w.show
    w.tick("Searching and loading tag data")

    metas.append start_meta
    processed_metas.append start_meta.class1 + ":" + start_meta.name
    for i as integer = 0 to start_meta.data.dependencies.ubound
        meta_queue.append start_meta.data.dependencies(i).tag_class _
            + ":" + start_meta.data.dependencies(i).tag_name
    next

```

```

for i as integer = 0 to start_meta.data.loneIDs.ubound
    meta_queue.append start_meta.data.loneIDs(i).tag_class _
    + ":" + start_meta.data.loneIDs(i).tag_name
next

//now to loop through the queue
while meta_queue.Ubound > -1
    if processed metas.IndexOf(meta_queue(0)) > - 1 then
        meta_queue.Remove(0)
        continue
    end
    dim temp_str() as string = meta_queue(0).split(":")
    if temp_str.ubound < 1 then
        meta_queue.Remove(0)
        continue
    end
    if temp_str(1) = "nulled out" then
        meta_queue.Remove(0)
        continue
    end

    meta_f = name_to_folderitem(meta_queue(0))
    if meta_f = nil then continue
    tag_name = meta_f.name.ReplaceAll(".meta","")
    xml_f = meta_f.Parent.Child(tag_name + ".xml")
    if not xml_f.Exists then
        continue
    end
    data_flag = false
    data_f = meta_f.Parent.Child(tag_name + " Data")
    if data_f.Exists and data_f.Directory then data_flag = true

    dim temp_meta as h1.meta
    if data_flag then
        temp_meta = h1.load_meta(meta_f, xml_f, data_f)
    else
        temp_meta = h1.load_meta(meta_f, xml_f)
    end

    metas.append temp_meta
    processed_metas.append temp_meta.class1 + ":" + temp_meta.name
    for i as integer = 0 to temp_meta.data.dependencies.ubound
        meta_queue.append temp_meta.data.dependencies(i).tag_class _
        + ":" + temp_meta.data.dependencies(i).tag_name
    next
    for i as integer = 0 to temp_meta.data.loneIDs.ubound
        meta_queue.append temp_meta.data.loneIDs(i).tag_class _
        + ":" + temp_meta.data.loneIDs(i).tag_name
    next
wend

dim temp_meta_pack as new H1.Meta_Pack
for i as integer = 0 to UBound(metas)
    temp_meta_pack.metas.append metas(i)

```

```

next
temp_meta_pack.class1 = metas(0).class1
temp_meta_pack.name = metas(0).name
main.extracted_meta_pack.append temp_meta_pack
main.extracted_meta_folders.item.append UBound(main.extracted_meta_pack)
main.update_list

w.close
MsgBox("Tags Loaded Successfully!")
End Sub

```

### **recursive\_load\_tag\_thread.init:**

```

Sub init(in_meta_f as folderItem, in_folder_f as folderItem, byref in_main as Main_Window)
    meta_f = in_meta_f
    folder_f = in_folder_f
    main = in_main
End Sub

```

### **recursive\_load\_tag\_thread.name\_to\_folderitem:**

```

Function name_to_folderitem(name as string) As folderItem
    dim top_split() as string = name.split(":")
    dim tag_class as string = top_split(0)
    dim filename as string = top_split(1)
    dim name_split() as string = filename.split("\")

    dim f as FolderItem = folder_f
    for i as integer = 0 to name_split.Ubound - 1
        dim folder_str as string = name_split(i)
        if folder_str = "" then folder_str = "BLANK"
        f = f.Child(folder_str)
        if not f.Exists then return nil
    next
    f = f.Child(name_split(name_split.Ubound) + "." + tag_class.reverse + ".meta")
    if not f.Exists then return nil
    return f
End Function

Private folder_f As folderItem

Private main As Main_Window

Private meta_f As folderItem

End Class

```

## **Class Preferences Window**

Inherits Window

### **Preferences\_Window.CloseWindow:**

```

Function CloseWindow() As Boolean
    self.Close

    Return True

```

End Function

### **Preferences\_Window.Constructor:**

Sub Constructor()

    // Calling the overridden superclass constructor.

    Super.Window

    //needs to do something here where it loads the current crop of Preferences

    current\_prefs = new H1.preferences

    dim f as FolderItem = GetFolderItem("").Child("Data")

    if not f.Exists then

        f.CreateAsFolder()

    end

    f = f.Child("preferences.xml")

    if f.Exists then

        current\_prefs.read(f)

    end

    change\_ok = false

    update\_controls

    change\_ok = true

End Sub

### **Preferences\_Window.update\_controls:**

Sub update\_controls()

    change\_ok = false

    if current\_prefs.byName then

        Radio\_name.Value = true

        Radio\_tagclass.Value = false

    else

        Radio\_name.Value = false

        Radio\_tagclass.Value = true

    end

    if current\_prefs.Ent\_over\_xml then

        Radio\_ent.Value = true

        Radio\_hmt.Value = false

    else

        Radio\_ent.Value = false

        Radio\_hmt.Value = true

    end

    if current\_prefs.explicit\_bitmap then

        Check\_explicit.Value = true

        edit\_explicit.Enabled = true

        edit\_explicit.Text = current\_prefs.explicit\_bitmap\_address

        push\_explicit.Enabled = true

    else

        Check\_explicit.Value = false

        edit\_explicit.Enabled = false

        edit\_explicit.text = ""

        push\_explicit.Enabled = false

    end



```

if current_prefs.scroll_edit then
    Radio_scroll.Value = true
    Radio_classic.Value = false
    Check_edit_by_offset.Enabled = true
else
    Radio_scroll.Value = false
    Radio_classic.Value = true
    Check_edit_by_offset.Enabled = false
end

if current_prefs.edit_by_offset then
    Check_edit_by_offset.Value = true
else
    Check_edit_by_offset.Value = false
end

check_show_hidden.Value = current_prefs.show_ent_hidden

list_plugins.DeleteAllRows
dim old_favored() as string = current_prefs.favored_plugins
dim favored() as string
dim f as FolderItem = GetFolderItem("").Child("Plugins")
//first remove non-existant plugin folders
for i as integer = 0 to old_favored.Ubound
    dim found as boolean = false
    for j as integer = 1 to f.count
        dim temp_f as FolderItem
        temp_f = f.item(j)
        if temp_f.Directory AND temp_f.Name = old_favored(i) then
            found = true
            exit for j
        end
    next
    if found then
        favored.Append(old_favored(i))
    end
next
//now add any new folders
for i as integer = 1 to f.count
    if f.item(i).Directory AND favored.IndexOf(f.item(i).name) = -1 then
        favored.Append f.item(i).Name
    end
next
current_prefs.favored_plugins = favored
for i as integer = 0 to UBound(favored)
    list_plugins.AddRow(favored(i))
next

change_ok = true
End Sub

```

### **Preferences\_Window.update\_windows:**

```
Sub update_windows()
```

```

dim favored() as string
for i as integer = 0 to list_plugins.ListCount-1
    favored.Append list_plugins.Cell(i,0)
next
current_prefs.favored_plugins = favored
dim f as FolderItem = GetFolderItem("").Child("Data")
if not f.Exists then
    f.CreateAsFolder()
end
f = f.child("preferences.xml")
current_prefs.write(f)
for i as integer = 0 to WindowCount-1
    if Window(i) isa Main_Window then
        Main_Window(Window(i)).update_preferences
    end
next
End Sub
change_ok As boolean

current_prefs As h1.preferences

```

## Preferences\_Window Note: The layout

The layout

Ok so here's what I plan to do with this:

plugins:

a list box for ordering which plugins are picked from first (in case a set doesn't have plugins)?

OR

a drop down menu for selecting which plugin folder in the folder "Plugins"

maybe something so you can have .xml and .ent in one folder (logic for finding which first)

a set of check boxes for things to display (like ignoring ints, enums, dependencies) ala entity

rendering:

deciding whether scnr uses: boxs, lod1, lod2 ect

deciding whether scnr uses bitmaps

deciding whether scnr uses mod2s with bitmaps

deciding whether mod2 uses bitmaps

deciding whether or not to use an explicit bitmap path

## Preferences\_Window Control push\_apply:

```

Sub Action()
    update_windows
    MsgBox("Preferences saved!")
    self.close
End Sub

```

## Preferences\_Window Control Radio\_tagclass:

```

Sub Action()

```

```

    Radio_name.Value = false
    current_prefs.byName = false
End Sub

```

### **Preferences\_Window Control Radio\_name:**

```

Sub Action()
    Radio_tagclass.Value = false
    current_prefs.byName = true
End Sub

```

### **Preferences\_Window Control push\_explicit:**

```

Sub Action()
    dim f as FolderItem
    f = GetOpenFolderItem(H1_Filetypes.HaloMapFile)
    if f = nil then return
    if f.Exists then
        current_prefs.explicit_bitmap_address = f.absolutepath
        current_prefs.explicit_bitmap = true
        edit_explicit.text = f.absolutepath
    end
End Sub

```

### **Preferences\_Window Control Check\_explicit:**

```

Sub Action()
    if not change_ok then Return
    if me.Value then
        edit_explicit.Enabled = true
        edit_explicit.text = current_prefs.explicit_bitmap_address
        push_explicit.Enabled = true
    else
        edit_explicit.Enabled = false
        edit_explicit.Text = ""
        push_explicit.Enabled = false
        current_prefs.explicit_bitmap = false
    end
End Sub

```

### **Preferences\_Window Control list\_plugins:**

```

Function DragReorderRows(newPosition as Integer, parentRow as Integer) As Boolean
    dim favored() as string
    for i as integer = 0 to list_plugins.ListCount-1
        favored.Append list_plugins.Cell(i,0)
    next
    current_prefs.favored_plugins = favored
End Function

```

### **Preferences\_Window Control Radio\_ent:**

```

Sub Action()
    Radio_hmt.Value = false
    current_prefs.ent_over_xml = true
End Sub

```

### **Preferences\_Window Control radio\_hmt:**

```

Sub Action()
    Radio_ent.Value = false
    current_prefs.ent_over_xml = false
End Sub

```

### **Preferences\_Window Control Radio\_scroll:**

```

Sub Action()
    radio_classic.Value = false
    current_prefs.scroll_edit = true
    Check_edit_by_offset.Enabled = true
End Sub

```

### **Preferences\_Window Control radio\_classic:**

```

Sub Action()
    Radio_scroll.Value = false
    current_prefs.scroll_edit = false
    Check_edit_by_offset.Enabled = false
End Sub

```

### **Preferences\_Window Control check\_show\_hidden:**

```

Sub Action()
    current_prefs.show_ent_hidden = me.Value
End Sub

```

### **Preferences\_Window Control Check\_edit\_by\_offset:**

```

Sub Action()
    current_prefs.edit_by_offset = me.Value
End Sub
End Class

```

## **Class Preferences\_Window1**

Inherits Window

### **Preferences\_Window1.CloseWindow:**

```

Function CloseWindow() As Boolean
    self.Close

```

```

    Return True

```

```

End Function

```

### **Preferences\_Window1.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window

    //needs to do something here where it loads the current crop of Preferences
    current_prefs = new H1.preferences
    dim f as FolderItem = GetFolderItem("").Child("Data")
    if not f.Exists then
        f.CreateAsFolder()
    end if

```

```

end
f = f.Child("preferences.xml")
if f.Exists then
    current_prefs.read(f)
end
change_ok = false
update_controls
change_ok = true
End Sub

```

### **Preferences\_Window1.update\_controls:**

```

Sub update_controls()
    change_ok = false

    if current_prefs.byName then
        Radio_name.Value = true
        Radio_tagclass.Value = false
    else
        Radio_name.Value = false
        Radio_tagclass.Value = true
    end

    if current_prefs.Ent_over_xml then
        Radio_ent.Value = true
        Radio_hmt.Value = false
    else
        Radio_ent.Value = false
        Radio_hmt.Value = true
    end

    if current_prefs.explicit_bitmap then
        Check_explicit.Value = true
        edit_explicit.Enabled = true
        edit_explicit.Text = current_prefs.explicit_bitmap_address
        push_explicit.Enabled = true
    else
        Check_explicit.Value = false
        edit_explicit.Enabled = false
        edit_explicit.text = ""
        push_explicit.Enabled = false
    end

    if current_prefs.scroll_edit then
        Radio_scroll.Value = true
        Radio_classic.Value = false
    else
        Radio_scroll.Value = false
        Radio_classic.Value = true
    end

    check_show_hidden.Value = current_prefs.show_ent_hidden
    Check_bsp_bitmap.value = current_prefs.bsp_bitmaps
    Check_bsp_model.value = current_prefs.bsp_models
    check_model_bitmap.value = current_prefs.model_bitmaps

```

```

list_plugins.DeleteAllRows
dim old_favored() as string = current_prefs.favored_plugins
dim favored() as string
dim f as FolderItem = GetFolderItem("").Child("Plugins")
//first remove non-existant plugin folders
for i as integer = 0 to old_favored.Ubound
    dim found as boolean = false
    for j as integer = 1 to f.count
        dim temp_f as FolderItem
        temp_f = f.item(j)
        if temp_f.Directory AND temp_f.Name = old_favored(i) then
            found = true
            exit for j
        end
    next
    if found then
        favored.Append(old_favored(i))
    end
next
//now add any new folders
for i as integer = 1 to f.count
    if f.item(i).Directory AND favored.IndexOf(f.item(i).Name) = -1 then
        favored.Append f.item(i).Name
    end
next
current_prefs.favored_plugins = favored
for i as integer = 0 to UBound(favored)
    list_plugins.AddRow(favored(i))
next

change_ok = true
End Sub

```

### **Preferences\_Window1.update\_windows:**

```

Sub update_windows()
    dim f as FolderItem = GetFolderItem("").Child("Data")
    if not f.Exists then
        f.CreateAsFolder()
    end
    f = f.child("preferences.xml")
    current_prefs.write(f)
    for i as integer = 0 to WindowCount-1
        if Window(i) isa Main_Window then
            Main_Window(Window(i)).update_preferences
        end
    next
End Sub

change_ok As boolean

current_prefs As h1.preferences

```

### **Preferences\_Window1 Note: The layout**

The layout

Ok so here's what I plan to do with this:

plugins:

a list box for ordering which plugins are picked from first (in case a set doesn't have plugins)?

OR

a drop down menu for selecting which plugin folder in the folder "Plugins"

maybe something so you can have .xml and .ent in one folder (logic for finding which first)

a set of check boxes for things to display (like ignoring ints, enums, dependencies) ala entity

rendering:

deciding whether scnr uses: boxs, lod1, lod2 ect

deciding whether scnr uses bitmaps

deciding whether scnr uses mod2s with bitmaps

deciding whether mod2 uses bitmaps

deciding whether or not to use an explicit bitmap path

**Preferences\_Window1 Control push\_apply:**

Sub Action()

    update\_windows

    MsgBox("Preferences saved!")

    self.close

End Sub

**Preferences\_Window1 Control Radio\_tagclass:**

Sub Action()

    Radio\_name.Value = false

    current\_prefs.byName = false

End Sub

**Preferences\_Window1 Control Radio\_name:**

Sub Action()

    Radio\_tagclass.Value = false

    current\_prefs.byName = true

End Sub

**Preferences\_Window1 Control push\_explicit:**

Sub Action()

    dim f as FolderItem

    f = GetOpenFolderItem(H1\_Filetypes.HaloMapFile)

    if f.Exists then

        current\_prefs.explicit\_bitmap\_address = f.absolutepath

        edit\_explicit.text = f.absolutepath

    end

End Sub

**Preferences\_Window1 Control Check\_explicit:**

Sub Action()

    if not change\_ok then Return

```

if me.Value then
    edit_explicit.Enabled = true
    edit_explicit.text = current_prefs.explicit_bitmap_address
    push_explicit.Enabled = true
else
    edit_explicit.Enabled = false
    edit_explicit.Text = ""
    push_explicit.Enabled = false
end
End Sub

```

### **Preferences\_Window1 Control list\_plugins:**

```

Function DragReorderRows(newPosition as Integer, parentRow as Integer) As Boolean
    dim favored() as string
    for i as integer = 0 to list_plugins.ListCount-1
        favored.Append list_plugins.Cell(i,0)
    next
    current_prefs.favored_plugins = favored
End Function

```

### **Preferences\_Window1 Control Radio\_ent:**

```

Sub Action()
    Radio_hmt.Value = false
    current_prefs.ent_over_xml = true
End Sub

```

### **Preferences\_Window1 Control radio\_hmt:**

```

Sub Action()
    Radio_ent.Value = false
    current_prefs.ent_over_xml = false
End Sub

```

### **Preferences\_Window1 Control Radio\_scroll:**

```

Sub Action()
    radio_classic.Value = false
    current_prefs.scroll_edit = true
End Sub

```

### **Preferences\_Window1 Control radio\_classic:**

```

Sub Action()
    Radio_scroll.Value = false
    current_prefs.scroll_edit = false
End Sub

```

### **Preferences\_Window1 Control Check\_bsp\_model:**

```

Sub Action()
    current_prefs.bsp_models = me.Value
End Sub

```

### **Preferences\_Window1 Control Check\_bsp\_bitmap:**

```

Sub Action()
    current_prefs.bsp_bitmaps = me.Value

```



End Sub

### **Preferences\_Window1 Control check\_model\_bitmap:**

Sub Action()

    current\_prefs.model\_bitmaps = me.Value

End Sub

### **Preferences\_Window1 Control check\_show\_hidden:**

Sub Action()

    current\_prefs.show\_ent\_hidden = me.Value

End Sub

End Class

## **Class About\_Window**

Inherits Window

### **About\_Window.CloseWindow:**

Function CloseWindow() As Boolean

    self.Close

    Return true

End Function

End Class

## **Class Map\_Expand\_Window**

Inherits Window

### **Map\_Expand\_Window.tick:**

Sub tick(value as integer)

    ProgressBar1.Value = value

    StaticText2.text = "Loading map into memory: " + str(value) + "%"

    //self.Refresh

End Sub

End Class

## **Class Map\_Rebuild\_Window**

Inherits Window

### **Map\_Rebuild\_Window.tick:**

Sub tick(status as string, value as integer)

    ProgressBar1.Value = value

    StaticText2.text = status + ": " + str(value) + "%"

    //self.Refresh

End Sub

End Class

## **Class Tag\_Search\_Window**

Inherits Window

### **Tag\_Search\_Window.CloseWindow:**

```
Function CloseWindow() As Boolean
    self.Close
End Function
```

### **Tag\_Search\_Window.search:**

```
Sub search(input as string)
    dim temp_str() as string = input.Split(" ")

    //data for normal maps
    dim window_index(-1) as integer
    dim maps_lite_index(-1) as integer
    dim tag_index(-1) as integer

    //data for expanded maps
    dim window_expanded_index(-1) as integer
    dim maps_expanded_index(-1) as integer
    dim tag_expanded_index(-1) as integer

    //really really really lazy search method but the only way to do it without fancy algos
    for i as integer = 0 to WindowCount - 1
        if Window(i) isa Main_Window then
            for j as integer = 0 to Main_Window(Window(i)).maps_lite.ubound
                for k as integer = 0 to Main_Window(Window(i)).maps_lite(j).tags.ubound
                    dim term_not_used as Boolean = false
                    for l as integer = 0 to temp_str.ubound
                        if all_terms.Value then
                            if (not (Main_Window(window(i)).maps_lite(j).tags(k).name.instr(temp_str(l)) > 0)) AND (not
                                (Main_Window(window(i)).maps_lite(j).tags(k).class1.reverse.instr(temp_str(l)) > 0)) then
                                term_not_used = true
                                exit for l
                            end
                        else
                            if ((Main_Window(window(i)).maps_lite(j).tags(k).name.instr(temp_str(l)) > 0)) OR
                                ((Main_Window(window(i)).maps_lite(j).tags(k).class1.reverse.instr(temp_str(l)) > 0)) then
                                window_index.append i
                                maps_lite_index.append j
                                tag_index.append k
                                exit for l
                            end
                        end
                    next
                    if all_terms.value and (not term_not_used) then
                        window_index.append i
                        maps_lite_index.append j
                        tag_index.append k
                    end
                next
            next
            for j as integer = 0 to Main_Window(Window(i)).maps_expanded.ubound
                for k as integer = 0 to Main_Window(Window(i)).maps_expanded(j).tags.ubound
                    dim term_not_used as Boolean = false
```

```

for l as integer = 0 to temp_str.ubound
    if all_terms.Value then
        if (not (Main_Window(window(i)).maps_expanded(j).tags(k).name.instr(temp_str(l)) > 0)) AND (not
            (Main_Window(window(i)).maps_expanded(j).tags(k).class1.reverse.instr(temp_str(l)) > 0)) then
            term_not_used = true
            exit for l
        end
    else
        if ((Main_Window(window(i)).maps_expanded(j).tags(k).name.instr(temp_str(l)) > 0)) OR
            ((Main_Window(window(i)).maps_expanded(j).tags(k).class1.reverse.instr(temp_str(l)) > 0)) then
            window_expanded_index.append i
            maps_expanded_index.append j
            tag_expanded_index.append k
            exit for l
        end
    end
next
if all_terms.value and (not term_not_used) then
    window_expanded_index.append i
    maps_expanded_index.append j
    tag_expanded_index.append k
end
next
end
next

```

```

//ok so now what?
//need some way to map the current window inds to an array that references those windows
redim w(-1)
dim temp_win_ar(-1) as integer
for i as integer = 0 to window_index.ubound
    temp_win_ar.append window_index(i)
next
for i as integer = 0 to window_expanded_index.ubound
    temp_win_ar.append window_expanded_index(i)
next
temp_win_ar = remove_redundant_ints(temp_win_ar)
for i as integer = 0 to UBound(temp_win_ar)
    w.Append(Main_Window(Window(temp_win_ar(i))))
next

```

```

Listbox1.DeleteAllRows
for i as integer = 0 to UBound(window_index)
    Listbox1.AddRow( Main_Window(Window(window_index(i))).maps_lite( maps_lite_index(i) ).fs.displayname )
    Listbox1.CellTag(Listbox1.LastIndex, 0) = "maps_lite:" + str(maps_lite_index(i))
    Listbox1.Cell(Listbox1.LastIndex, 1) =
        Main_Window(Window(window_index(i))).maps_lite( maps_lite_index(i) ).tags( tag_index(i) ).class1.reverse
    Listbox1.CellTag(Listbox1.LastIndex, 1) =
        Main_Window(Window(window_index(i))).maps_lite( maps_lite_index(i) ).tags( tag_index(i) ).class1.reverse
    Listbox1.Cell(Listbox1.LastIndex, 2) =
        Main_Window(Window(window_index(i))).maps_lite( maps_lite_index(i) ).tags( tag_index(i) ).name
    Listbox1.CellTag(Listbox1.LastIndex, 2) =
        Main_Window(Window(window_index(i))).maps_lite( maps_lite_index(i) ).tags( tag_index(i) ).name
    Listbox1.Cell(Listbox1.LastIndex, 3) = "Window Index"

```

```

if temp_win_ar.IndexOf( window_index(i) ) >= 0 then
    Listbox1.CellTag( Listbox1.LastIndex, 3) = temp_win_ar.IndexOf( window_index(i) )
else
    Listbox1.RemoveRow(Listbox1.LastIndex) //this should never occur
    break
end
next
for i as integer = 0 to UBound(window_expanded_index)
    Listbox1.AddRow( Main_Window(Window(window_expanded_index(i))).maps_expanded( maps_expanded_index(i) ).fs.displayname )
    Listbox1.CellTag(Listbox1.LastIndex, 0) = "maps_expanded:" + str(maps_expanded_index(i))
    Listbox1.Cell(Listbox1.LastIndex, 1) =
    Main_Window(Window(window_expanded_index(i))).maps_expanded( maps_expanded_index(i) ).tags( tag_expanded_index(i) ).class1.reverse
    Listbox1.CellTag(Listbox1.LastIndex, 1) =
    Main_Window(Window(window_expanded_index(i))).maps_expanded( maps_expanded_index(i) ).tags( tag_expanded_index(i) ).class1.reverse
    Listbox1.Cell(Listbox1.LastIndex, 2) =
    Main_Window(Window(window_expanded_index(i))).maps_expanded( maps_expanded_index(i) ).tags( tag_expanded_index(i) ).name
    Listbox1.CellTag(Listbox1.LastIndex, 2) =
    Main_Window(Window(window_expanded_index(i))).maps_expanded( maps_expanded_index(i) ).tags( tag_expanded_index(i) ).name
    Listbox1.Cell(Listbox1.LastIndex, 3) = "Window Index"
    if temp_win_ar.IndexOf( window_expanded_index(i) ) >= 0 then
        Listbox1.CellTag( Listbox1.LastIndex, 3) = temp_win_ar.IndexOf( window_expanded_index(i) )
    else
        Listbox1.RemoveRow(Listbox1.LastIndex) //this should never occur
        break
    end
next
End Sub
w() As Main_Window

```

### **Tag\_Search\_Window Control EditField1:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        PushButton1.Push
    end
End Function

```

### **Tag\_Search\_Window Control PushButton1:**

```

Sub Action()
    search(EditField1.text)
End Sub

```

### **Tag\_Search\_Window Control ListBox1:**

```

Sub DoubleClick()
    Dim row as Integer
    row= Me.RowFromXY( System.MouseX - Me.Left - Self.Left,_
    System.MouseY - Me.Top - Self.Top)

    dim map_index as string

```

```

dim class2 as string
dim fullname as string
dim window_index as integer

map_index = me.CellTag(row, 0)
class2 = me.CellTag(row, 1)
fullname = me.CellTag(row, 2)
window_index = me.CellTag(row, 3)

if w(window_index).select_tag(map_index, class2, fullname) then self.Close
End Sub
End Class

```

## **Class PluginLibUniversal**

### **PluginLibUniversal.refresh\_as\_Ent:**

Sub refresh\_as\_Ent(f as folderitem, is\_reflexive as boolean, in\_this\_reflexive\_index() as integer, show\_hidden as boolean = true)

```

//reflexive_index info:
//1 = the main struct
//>1 = any of the listed reflexive's

```

```

dim xdoc as XmlDocument
dim xdoc2 as XmlNode
dim node as XmlNode
dim count, count2 as integer
redim this_reflexive_index(-1)

```

```

for i as integer = 0 to ubound(in_this_reflexive_index)
    this_reflexive_index.append(in_this_reflexive_index(i))
next

```

```

if not f.Exists then
    no_plugin = true
    return
end

```

```

xdoc = new XmlDocument
xdoc.LoadXml f
this_plugin_name = xdoc.DocumentElement.GetAttribute("class")
if not is_reflexive then

```

```

count = xdoc.DocumentElement.childcount
this_reflexive_name = "main"
this_reflexive_size = val(xdoc.DocumentElement.GetAttribute("headersize"))
this_reflexive_offset = 0
else
  //deal with recursive reflexives
  dim struct_count as integer = 0//1
  count = xdoc.DocumentElement.childcount
  for i as integer = 0 to count-1
    node = xdoc.DocumentElement.Child(i)
    if node.name = "struct" then
      struct_count = struct_count+1
    end
    if struct_count = this_reflexive_index(0) then
      xdoc2=xdoc.DocumentElement.child(i)
      is_reflexive = true
      this_reflexive_name = node.GetAttribute("name")
      this_reflexive_size = val(node.GetAttribute("size"))
      this_reflexive_offset = val(node.GetAttribute("offset"))
      count = xdoc2.ChildCount
      i = count + 1
    end
  next

  for j as integer = 1 to ubound(this_reflexive_index)
    struct_count = 0
    for i as integer = 0 to count-1
      node = xdoc2.child(i)
      if node.name = "struct" then
        struct_count = struct_count+1
      end
      if struct_count = this_reflexive_index(j) then
        xdoc2=xdoc2.child(i)
        is_reflexive = true
        this_reflexive_name = node.GetAttribute("name")
        this_reflexive_size = val(node.GetAttribute("size"))
        count = xdoc2.ChildCount
        i = count + 1
      end
    next
  next
end

redim bitmask16_data(-1)
redim bitmask16_name(-1)
redim bitmask16_offset(-1)
redim bitmask16_order(-1)
redim bitmask32_data(-1)
redim bitmask32_name(-1)
redim bitmask32_offset(-1)
redim bitmask32_order(-1)
redim bitmask8_data(-1)
redim bitmask8_name(-1)
redim bitmask8_offset(-1)

```

redim bitmask8\_order(-1)  
redim colorARGB\_name(-1)  
redim colorARGB\_offset(-1)  
redim colorARGB\_order(-1)  
redim colorbyte\_name(-1)  
redim colorbyte\_offset(-1)  
redim colorbyte\_order(-1)  
redim colorRGB\_name(-1)  
redim colorRGB\_offset(-1)  
redim colorRGB\_order(-1)  
redim dependency\_name(-1)  
redim dependency\_offset(-1)  
redim dependency\_order(-1)  
redim double\_name(-1)  
redim double\_offset(-1)  
redim double\_order(-1)  
redim float\_name(-1)  
redim float\_offset(-1)  
redim float\_order(-1)  
redim id16\_data(-1)  
redim id16\_name(-1)  
redim id16\_offset(-1)  
redim id16\_order(-1)  
redim id32\_data(-1)  
redim id32\_name(-1)  
redim id32\_offset(-1)  
redim id32\_order(-1)  
redim id8\_data(-1)  
redim id8\_name(-1)  
redim id8\_offset(-1)  
redim id8\_order(-1)  
redim index\_name(-1)  
redim index\_offset(-1)  
redim index\_data(-1)  
redim index\_order(-1)  
redim int16\_name(-1)  
redim int16\_offset(-1)  
redim int16\_order(-1)  
redim int32\_name(-1)  
redim int32\_offset(-1)  
redim int32\_order(-1)  
redim int8\_name(-1)  
redim int8\_offset(-1)  
redim int8\_order(-1)  
redim loneID\_name(-1)  
redim loneID\_offset(-1)  
redim loneID\_order(-1)  
redim reflexives(-1)  
redim reflexive\_name(-1)  
redim reflexive\_offset(-1)  
redim reflexive\_name\_offset(-1)  
redim reflexive\_order(-1)  
redim string128\_name(-1)  
redim string128\_offset(-1)

```

redim string128_order(-1)
redim string32_name(-1)
redim string32_offset(-1)
redim string32_order(-1)
redim string4_name(-1)
redim string4_offset(-1)
redim string4_order(-1)

for i as integer = 0 to count - 1
    if not is_reflexive then
        node = xdoc.DocumentElement.Child(i)
    else
        node = xdoc2.child(i)
    end if

    dim type as string = node.name

    dim visible as boolean
    try
        if node.GetAttribute("visible").Uppercase = "TRUE" then
            visible = true
        else
            visible = false
        end
    catch
        visible = false
    end try
    if show_hidden then visible = true

    //get the number of nodes in the <value> tag
    count2 = node.childcount

    //get the actual offset value first
    dim offset as integer
    dim nil_flag as boolean = false
    //check for hex format
    try
        if (left(node.GetAttribute("offset"), 2) = "0x") then
            try
                offset = val("&h" + mid(node.GetAttribute("offset"), 3))
            catch err1 as NilObjectException
                nil_flag = true
            end try
        else
            //otherwise it's normal
            try
                offset = val(node.GetAttribute("offset"))
            catch
                nil_flag = true
            end try
        end
    catch
        nil_flag = true

```



```

end try

dim name as string
try
    name = node.GetAttribute("name")
catch
    name = "Unknown"
end try

if nil_flag = true then
    type = "No offset"
end

if visible or Show_Hidden then
    select case type //the <type> tag's value
        'node.child(1).firstChild.value //the value in <name>
        'name //the value in <offset>

    case "bitmask16"
        bitmask16_offset().append offset
        bitmask16_name().append name
        bitmask16_order.Append i

    case "bitmask32"
        bitmask32_offset().append offset //the value in <name>
        bitmask32_name().append name//the value in <offset>
        bitmask32_order.Append i

    case "bitmask8"
        bitmask8_offset().append offset
        bitmask8_name().append name
        bitmask8_order.Append i

    case "colorARGB"
        colorARGB_offset().append offset
        colorARGB_name().append name
        colorARGB_order.Append i

    case "colorbyte"
        colorbyte_offset().append offset
        colorbyte_name().append name
        colorbyte_order.Append i

    case "colorRGB"
        colorRGB_offset().append offset
        colorRGB_name().append name
        colorRGB_order.Append i

    case "dependency"
        dependency_offset.Append offset
        dependency_name.Append name
        dependency_order.Append i

```

```

case "double"
    double_offset().append offset
    double_name().append name
    double_order.Append i

case "float"
    float_offset().append offset //the value in <name>
    float_name().append name //the value in <offset>
    float_order.Append i

case "enum16"
    id16_offset().append offset //the value in <name>
    id16_name().append name
    id16_order.Append i

case "enum32"
    id32_offset().append offset //the value in <name>
    id32_name().append name
    id32_order.Append i

case "enum8"
    id8_offset().append offset
    id8_name().append name
    id8_order.Append i

case "index"
    index_offset().append offset
    index_name().append name
    index_order.Append i

case "int16"
    int16_offset().append offset //the value in <name>
    int16_name().append name //the value in <offset>
    int16_order.Append i

case "short"
    int16_offset().append offset //the value in <name>
    int16_name().append name //the value in <offset>
    int16_order.Append i

case "int32"
    int32_offset.Append offset
    int32_name.Append name
    int32_order.Append i

case "long"
    int32_offset.Append offset
    int32_name.Append name
    int32_order.Append i

case "int8"
    int8_offset.Append offset
    int8_name.Append name
    int8_order.Append i

```

```

case "char"
    int8_offset.Append offset
    int8_name.Append name
    int8_order.Append i

case "lonelD"
    lonelD_offset.Append offset
    lonelD_name.Append name
    lonelD_order.Append i

case "struct" //aka reflexive

    //new section to add extra data
    dim name_offset as integer = -1
    dim name_node as boolean = false
    try
        if (left(node.GetAttribute("name_offset"), 2) = "0x") then
            try
                name_offset = val("&h" + mid(node.GetAttribute("name_offset"), 3))
                if node.GetAttribute("name_offset") = "" then
                    name_offset = -1
                end
            catch err1 as NilObjectException
                name_offset = -1
            end try
        else
            //otherwise it's normal
            try
                name_offset = val(node.GetAttribute("name_offset"))
                if node.GetAttribute("name_offset") = "" then
                    name_offset = -1
                end
            catch err1 as NilObjectException
                name_offset = -1
            end try
        end
    catch
        name_offset = -1
    end try

    //and because conure is a flaming piece of doodoo
    'try
    'dim node_ind as integer = val(node.GetAttribute("nameNode"))
    'name_node = true
    'catch err1 as NilObjectException
    'end try
    //end new section

    reflexive_offset.append offset
    reflexive_name.append name
    reflexive_order.Append i
    dim ent as new PluginLibUniversal

```

```

    dim ref_index() as integer
    for k as integer = 0 to ubound(this_reflexive_index)
        ref_index.append this_reflexive_index(k)
    next
    ref_index.append ubound(reflexive_name) + 1
    //break
    ent.refresh_as_Ent(f, true, ref_index)
    reflexives.append ent
    //may need to do some silly things here
    reflexive_name_offset.Append name_offset

case "string128"
    string128_offset().append offset
    string128_name().append name
    string128_order.Append i

case "string32"
    string32_offset().append offset
    string32_name().append name
    string32_order.Append i

case "string4"
    string4_offset().append offset
    string4_name().append name
    string4_order.Append i

end select

//bitmask16s
if(type = "bitmask16")then
    dim temp1 as new data
    For j as integer = 0 to count2-1
        dim index as integer
        dim info as string
        try
            index = val(node.child(j).GetAttribute("value"))
            info = node.child(j).GetAttribute("name")
        catch
            index = -1
            info = "null"
        end try
        if index > -1 then
            temp1.values.append index+1 //lazy ftw!
            temp1.labels.append info
        end
    next
    bitmask16_data().append temp1
end

//bitmask32s
if(type = "bitmask32")then
    dim temp1 as new data
    For j as integer = 0 to count2-1
        dim index as integer

```

```

    dim info as string
    try
        index = val(node.child(j).GetAttribute("value"))
        info = node.child(j).GetAttribute("name")
    catch
        index = -1
        info = "null"
    end try
    if index > -1 then
        temp1.values.append index+1
        temp1.labels.append info
    end
next
bitmask32_data().append temp1
end

```

```

//bitmask8s
if(type = "bitmask8")then
    dim temp1 as new data
    For j as integer = 0 to count2-1
        dim index as integer
        dim info as string
        try
            index = val(node.child(j).GetAttribute("value"))
            info = node.child(j).GetAttribute("name")
        catch
            index = -1
            info = "null"
        end try
        if index > -1 then
            temp1.values.append index+1
            temp1.labels.append info
        end
    next
    bitmask8_data().append temp1
end

```

```

//handle id16s
if(type = "enum16")then
    dim temp2 as new data
    dim info as string
    dim value as int16
    For j as integer = 0 to count2-1
        dim err as boolean = false
        try
            dim temp as xmlNode = node.Child(j)
            value = val(node.child(j).GetAttribute("value"))
            info = node.child(j).GetAttribute("name")
        catch
            err = true
        end try
        if err = false then
            temp2.values.append value
            temp2.labels.append info
        end
    next
    enum16_data().append temp2
end

```

```

        end
    next
    id16_data().append temp2
end

```

```

//handle id32s
if(type = "enum32")then
    dim temp2 as new data
    dim info as string
    dim value as integer
    For j as integer = 0 to count2-1
        dim err as boolean = false
        try
            value = val(node.child(j).GetAttribute("value"))
            info = node.child(j).GetAttribute("name")
        catch
            err = true
        end try
        if err = false then
            temp2.values.append value
            temp2.labels.append info
        end
    next
    id32_data().append temp2
end

```

```

//handle id8s
if(type = "enum8")then
    dim temp2 as new data
    dim info as string
    dim value as int8
    For j as integer = 0 to count2-1
        dim err as boolean = false
        try
            value = val(node.child(j).GetAttribute("value"))
            info = node.child(j).GetAttribute("name")
        catch
            err = true
        end try
        if err = false then
            temp2.values.append value
            temp2.labels.append info
        end
    next
    id8_data().append temp2
end

```

```

if(type = "index")then
    dim temp2 as string
    temp2 = node.GetAttribute("reflexive")
    index_data.Append temp2
end

```

```

end

```

next

Exception err

break

//this is just here to handle bad plugin access

End Sub

### **PluginLibUniversal.refresh\_as\_HMT:**

Sub refresh\_as\_HMT(f as folderitem, is\_reflexive as boolean, in\_this\_reflexive\_index() as integer, show\_hidden as boolean = true)

//reflexive\_index info:

//1 = the main struct

//>1 = any of the listed reflexives in the <plugin> struct

//<0 = any of the reflexives that are stored inside of a <struct> struct

dim xdoc as XmlDocument

dim xdoc2 as XmlNode

dim node as XmlNode

dim count, count2 as integer

redim this\_reflexive\_index(-1)

for i as integer = 0 to ubound(in\_this\_reflexive\_index)

    this\_reflexive\_index.append(in\_this\_reflexive\_index(i))

next

if not f.Exists then

    no\_plugin = true

    return

end

xdoc = new XmlDocument

xdoc.LoadXml f

this\_plugin\_name = xdoc.DocumentElement.child(0).FirstChild.value

if not is\_reflexive then

    count = xdoc.DocumentElement.child(1).childcount

    this\_reflexive\_name = "main"

    this\_reflexive\_size = val(xdoc.DocumentElement.child(1).child(1).FirstChild.Value)

    this\_reflexive\_offset = 0

else

    //deal with recursive reflexives

    //first search through the highest level reflexive

    dim struct\_count as integer = 0

    if this\_reflexive\_index(0) > 0 then

        //this'll search for all structs in the <plugin> tag

        count = xdoc.DocumentElement.childcount

        for i as integer = 0 to count-1

            node = xdoc.DocumentElement.Child(i)

            if node.name = "struct" then

                struct\_count = struct\_count+1

            end

```

    if struct_count = this_reflexive_index(0) then
        xdoc2=xdoc.DocumentElement.child(i)
        is_reflexive = true
        count = xdoc2.ChildCount
        this_reflexive_name = xdoc2.child(0).FirstChild.value
        this_reflexive_size = val(xdoc2.child(1).FirstChild.value)
        i = count + 1
    end
next

else
    //this'll search for all structs inside of the main struct
    count = xdoc.DocumentElement.child(1).childcount
    for i as integer = 0 to count-1
        node = xdoc.DocumentElement.Child(1).child(i)

        if node.name = "struct" then
            struct_count = struct_count + 1
        end
        if struct_count = abs(this_reflexive_index(0))then
            xdoc2=xdoc.DocumentElement.child(1).child(i)
            is_reflexive = true
            count = xdoc2.ChildCount
            this_reflexive_name = xdoc2.child(0).FirstChild.value
            this_reflexive_size = val(xdoc2.child(1).FirstChild.value)

            i = count + 1
        end
    next
end

for j as integer = 1 to ubound(this_reflexive_index)
    struct_count = 0

    if (this_reflexive_index(j) > 0) then
        //all structs outside of main
        count = xdoc.DocumentElement.childcount

        for i as integer = 0 to count-1
            node = xdoc.DocumentElement.Child(i)
            if node.name = "struct" then
                struct_count = struct_count+1
            end
            if struct_count = this_reflexive_index(j) then
                xdoc2=xdoc.DocumentElement.child(i)
                is_reflexive = true
                count = xdoc2.ChildCount
                this_reflexive_name = xdoc2.child(0).FirstChild.value
                this_reflexive_size = val(xdoc2.child(1).FirstChild.value)

                i = count + 1
            end
        next
    end
next

```



```

else

    //internal structs
    for i as integer = 0 to count-1
        node = xdoc2.child(i)

        if node.name = "struct" then
            struct_count = struct_count + 1
        end
        if struct_count = abs(this_reflexive_index(0))then
            xdoc2=xdoc2.child(i)
            is_reflexive = true
            count = xdoc2.ChildCount
            this_reflexive_name = xdoc2.child(0).FirstChild.value
            this_reflexive_size = val(xdoc2.child(1).FirstChild.value)

            i = count + 1
        end
    next
end
next

end

redim bitmask16_data(-1)
redim bitmask16_name(-1)
redim bitmask16_offset(-1)
redim bitmask16_order(-1)
redim bitmask32_data(-1)
redim bitmask32_name(-1)
redim bitmask32_offset(-1)
redim bitmask32_order(-1)
redim bitmask8_data(-1)
redim bitmask8_name(-1)
redim bitmask8_offset(-1)
redim bitmask8_order(-1)
redim colorARGB_name(-1)
redim colorARGB_offset(-1)
redim colorARGB_order(-1)
redim colorbyte_name(-1)
redim colorbyte_offset(-1)
redim colorbyte_order(-1)
redim colorRGB_name(-1)
redim colorRGB_offset(-1)
redim colorRGB_order(-1)
redim dependency_name(-1)
redim dependency_offset(-1)
redim dependency_order(-1)
redim double_name(-1)
redim double_offset(-1)
redim double_order(-1)
redim float_name(-1)
redim float_offset(-1)

```

```

redim float_order(-1)
redim id16_data(-1)
redim id16_name(-1)
redim id16_offset(-1)
redim id16_order(-1)
redim id32_data(-1)
redim id32_name(-1)
redim id32_offset(-1)
redim id32_order(-1)
redim id8_data(-1)
redim id8_name(-1)
redim id8_offset(-1)
redim id8_order(-1)
redim index_name(-1)
redim index_offset(-1)
redim index_data(-1)
redim index_order(-1)
redim int16_name(-1)
redim int16_offset(-1)
redim int16_order(-1)
redim int32_name(-1)
redim int32_offset(-1)
redim int32_order(-1)
redim int8_name(-1)
redim int8_offset(-1)
redim int8_order(-1)
redim lonelD_name(-1)
redim lonelD_offset(-1)
redim lonelD_order(-1)
redim reflexives(-1)
redim reflexive_name(-1)
redim reflexive_offset(-1)
redim reflexive_name_offset(-1)
redim reflexive_order(-1)
redim string128_name(-1)
redim string128_offset(-1)
redim string128_order(-1)
redim string32_name(-1)
redim string32_offset(-1)
redim string32_order(-1)
redim string4_name(-1)
redim string4_offset(-1)
redim string4_order(-1)

for i as integer = 2 to count - 1
    if not is_reflexive then
        node = xdoc.DocumentElement.Child(1).Child(i)
    else
        node = xdoc2.child(i)
    end if

    //get the number of nodes in the <value> tag
    count2 = node.childcount

```

```

//get the actual offset value first
dim nil_flag as boolean = false
dim temp as string
try
    if (left(node.child(1).firstChild.value, 2) = "0x") then
        try
            temp = "&h" + mid(node.child(1).firstChild.value, 3)
        catch err1 as NilObjectException
            nil_flag = true
        end try
    else
        try
            temp = "&h" + node.child(1).firstChild.value
        catch err2 as NilObjectException
            nil_flag = true
        end try
    end
catch
    nil_flag = true
end try

dim type as string
try
    type = node.child(0).firstChild.value
catch err3 as NilObjectException
    type = "Unknown"
end try

dim name as string
try
    name = node.child(2).firstChild.value
catch err4 as NilObjectException
    name = "Unknown"
end try

if nil_flag = true then
    type = "No offset"
end

select case type //the <type> tag's value
    'node.child(1).firstChild.value //the value in <name>
    'name //the value in <offset>

case "bitmask16"
    bitmask16_offset().append val(temp)
    bitmask16_name().append name
    bitmask16_order.Append i

case "bitmask32"
    bitmask32_offset().append val(temp) //the value in <name>
    bitmask32_name().append name//the value in <offset>
    bitmask32_order.Append i

```

```

case "bitmask8"
    bitmask8_offset().append val(temp)
    bitmask8_name().append name
    bitmask8_order.Append i

case "colorARGB"
    colorARGB_offset().append val(temp)
    colorARGB_name().append name
    colorARGB_order.Append i

case "colorbyte"
    colorbyte_offset().append val(temp)
    colorbyte_name().append name
    colorbyte_order.Append i

case "colorRGB"
    colorRGB_offset().append val(temp)
    colorRGB_name().append name
    colorRGB_order.Append i

case "dependency"
    dependency_offset.append val(temp)
    dependency_name.Append name
    dependency_order.Append i

case "double"
    double_offset().append val(temp)
    double_name().append name
    double_order.Append i

case "float"
    float_offset().append val(temp) //the value in <name>
    float_name().append name //the value in <offset>
    float_order.Append i

case "id16"
    id16_offset().append val(temp) //the value in <name>
    id16_name().append name
    id16_order.Append i

case "id32"
    id32_offset().append val(temp) //the value in <name>
    id32_name().append name
    id32_order.Append i

case "id8"
    id8_offset().append val(temp)
    id8_name().append name
    id8_order.Append i

case "index"
    index_offset().append val(temp)
    index_name().append name
    index_order.Append i

```

```

case "int16"
    int16_offset().append val(temp) //the value in <name>
    int16_name().append name //the value in <offset>
    int16_order.Append i

case "short"
    int16_offset().append val(temp) //the value in <name>
    int16_name().append name //the value in <offset>
    int16_order.Append i

case "int32"
    int32_offset.append val(temp)
    int32_name.Append name
    int32_order.Append i

case "long"
    int32_offset.append val(temp)
    int32_name.Append name
    int32_order.Append i

case "int8"
    int8_offset.append val(temp)
    int8_name.Append name
    int8_order.Append i

case "char"
    int8_offset.append val(temp)
    int8_name.Append name
    int8_order.Append i

case "lonelD"
    lonelD_offset.append val(temp)
    lonelD_name.Append name
    lonelD_order.Append i

case "reflexive"
    reflexive_offset().append val(temp) //the value in <name>
    reflexive_name().append name
    reflexive_name_offset().append -1
    reflexive_order.Append i

case "string128"
    string128_offset().append val(temp)
    string128_name().append name
    string128_order.Append i

case "string32"
    string32_offset().append val(temp)
    string32_name().append name
    string32_order.Append i

case "string4"
    string4_offset().append val(temp)

```

```

string4_name().append name
string4_order.Append i

end select

'node.child(1).firstChild.value //the value in <name>
'name //the value in <offset>

//handle bitmask16s
if(type = "bitmask16")then
    dim temp1 as new data
    For j as integer = 3 to count2-1
        dim index as integer
        dim info as string
        try
            index = val(node.child(j).child(0).firstchild.value) //the value in <bit>
            info = node.child(j).child(1).firstchild.value //the value in <name>
        catch
            index = 0
            info = "null"
        end try
        if index > 0 then
            temp1.values.append index
            temp1.labels.append info
        end
    next
    bitmask16_data().append temp1
end

//handle bitmask32s
if(type = "bitmask32")then
    dim temp1 as new data
    For j as integer = 3 to count2-1
        dim index as integer
        dim info as string
        try
            index = val(node.child(j).child(0).firstchild.value) //the value in <bit>
            info = node.child(j).child(1).firstchild.value //the value in <name>
        catch
            index = 0
            info = "null"
        end try
        if index > 0 then
            temp1.values.append index
            temp1.labels.append info
        end
    next
    bitmask32_data().append temp1
end

//handle bitmask8s
if(type = "bitmask8")then
    dim temp1 as new data

```

```

For j as integer = 3 to count2-1
    dim index as integer
    dim info as string
    try
        index = val(node.child(j).child(0).firstchild.value) //the value in <bit>
        info = node.child(j).child(1).firstchild.value //the value in <name>
    catch
        index = 0
        info = "null"
    end try
    if index > 0 then
        temp1.values.append index
        temp1.labels.append info
    end
next
bitmask8_data().append temp1
end

//handle id16s
if(type = "id16")then
    dim temp2 as new data
    dim info as string
    dim value as int16
    For j as integer = 3 to count2-1
        dim err as boolean = false
        try
            value = val(node.child(j).child(0).firstchild.value) //the value in <value>
            info = node.child(j).child(1).firstchild.value //the value in <name>
        catch
            err = true
        end try
        if err = false then
            temp2.values.append( value )
            temp2.labels.append info
        end
    next
    id16_data().append temp2
end

//handle id32s
if(type = "id32")then
    dim temp2 as new data
    dim info as string
    dim value as integer
    For j as integer = 3 to count2-1
        dim err as boolean = false
        try
            value = val(node.child(j).child(0).firstchild.value) //the value in <value>
            info = node.child(j).child(1).firstchild.value //the value in <name>
        catch
            err = true
        end try
        if err = false then
            temp2.values.append( value )

```

```

        temp2.labels.append info
    end
next
id32_data().append temp2
end

//handle id8s
if(type = "id8")then
    dim temp2 as new data
    dim info as string
    dim value as int8
    For j as integer = 3 to count2-1
        dim err as boolean = false
        try
            value = val(node.child(j).child(0).firstchild.value) //the value in <value>
            info = node.child(j).child(1).firstchild.value //the value in <name>
        catch
            err = true
        end try
        if err = false then
            temp2.values.append( value )
            temp2.labels.append info
        end
    next
    id8_data().append temp2
end

if(type = "index") then
    dim temp2 as string
    temp2 = node.child(3).firstChild.value
    index_data.Append temp2
end

if node.name = "struct" then
    //break
    //this gets all the reflexives, but we'll have to store them in the same order as the reflexives tags
    dim temp2 as new PluginLibUniversal
    dim index2 as integer = -1*(ubound(reflexives)+2)
    dim ref_index(-1) as integer
    for j as integer = 0 to ubound(this_reflexive_index)
        ref_index.append(this_reflexive_index(j))
    next
    ref_index.append(index2)
    temp2.refresh_as_HMT(f, true, ref_index)
    reflexives.Append(temp2)
end

next

//more to come

//now search through all structs outside the main, and check to see if they're the relevant ones.

//still needs work on determining the correct struct. has repeated reading of "Clip"

```



```

dim last_index as integer = 0
dim struct_count as integer = 0
for i as integer = ubound(reflexives)+1 to ubound(reflexive_name)
    dim temp2 as new PluginLibUniversal
    dim ref_index(-1) as integer

    for j as integer = 0 to ubound(this_reflexive_index)
        ref_index.Append(this_reflexive_index(j))
    next

    count = xdoc.DocumentElement.ChildCount
    for j as integer = last_index to count-1
        node = xdoc.DocumentElement.Child(j)
        if node.name = "struct" then
            struct_count = struct_count+1
            for k as integer = 0 to ubound(reflexive_name)
                if node.child(0).FirstChild.value = reflexive_name(k) then
                    ref_index.append(struct_count)
                    temp2.refresh_as_HMT(f, true, ref_index)
                    reflexives.Append(temp2)
                    k = ubound(reflexive_name)
                    last_index = j+1
                    j=count
                end
            next
        end
    next
end
next
next

//finally, sort through the structs and arrange them in the same order as the reflexive
dim refs(-1) as PluginLibUniversal
for i as integer = 0 to ubound(reflexives)
    refs.append(reflexives(i))
next
redim reflexives(-1)
for i as integer = 0 to ubound(reflexive_name)
    for j as integer = 0 to ubound(refs)
        if(refs(j).this_reflexive_name = reflexive_name(i)) then
            refs(j).this_reflexive_offset = reflexive_offset(i) //retroactively apply the offset to the reflexive
            reflexives.append(refs(j))
            j = ubound(refs)+1
        end
    next
next
End Sub

bitmask16_data() As data

bitmask16_name() As string

bitmask16_offset() As Integer

bitmask16_order() As Integer

bitmask32_data() As data

```

bitmask32\_name() As string  
bitmask32\_offset() As Integer  
bitmask32\_order() As Integer  
bitmask8\_data() As data  
bitmask8\_name() As string  
bitmask8\_offset() As Integer  
bitmask8\_order() As Integer  
colorARGB\_name() As string  
colorARGB\_offset() As Integer  
colorARGB\_order() As Integer  
colorbyte\_name() As string  
colorbyte\_offset() As Integer  
colorbyte\_order() As Integer  
colorRGB\_name() As string  
colorRGB\_offset() As Integer  
colorRGB\_order() As Integer  
dependency\_name() As string  
dependency\_offset() As Integer  
dependency\_order() As Integer  
double\_name() As string  
double\_offset() As Integer  
double\_order() As Integer  
float\_name() As string  
float\_offset() As Integer  
float\_order() As Integer  
id16\_data() As data  
id16\_name() As string

id16\_offset() As Integer  
id16\_order() As Integer  
id32\_data() As data  
id32\_name() As string  
id32\_offset() As Integer  
id32\_order() As Integer  
id8\_data() As data  
id8\_name() As string  
id8\_offset() As integer  
id8\_order() As Integer  
index\_data() As string  
index\_name() As string  
index\_offset() As Integer  
index\_order() As Integer  
int16\_name() As string  
int16\_offset() As Integer  
int16\_order() As Integer  
int32\_name() As string  
int32\_offset() As Integer  
int32\_order() As Integer  
int8\_name() As string  
int8\_offset() As Integer  
int8\_order() As Integer  
lonelD\_name() As string  
lonelD\_offset() As Integer  
lonelD\_order() As Integer  
no\_plugin As boolean

reflexives(-1) As PluginLibUniversal  
reflexive\_name() As string  
reflexive\_name\_offset() As Integer  
reflexive\_offset() As Integer  
reflexive\_order() As Integer  
string128\_name() As string  
string128\_offset() As Integer  
string128\_order() As Integer  
string32\_name() As string  
string32\_offset() As Integer  
string32\_order() As Integer  
string4\_name() As string  
string4\_offset() As Integer  
string4\_order() As Integer  
this\_plugin\_name As string  
this\_reflexive\_index(-1) As Integer  
this\_reflexive\_name As string  
this\_reflexive\_offset As Integer  
this\_reflexive\_size As Integer  
End Class

## **Class data**

labels(-1) As string  
values(-1) As variant  
End Class

Class bitm\_image\_info

### **bitm\_image\_info.read:**

Sub read(byref br as binaryStream)

br.LittleEndian = true

position = br.Position

class1 = br.Read(4)

width = br.ReadShort

height = br.ReadShort

depth = br.ReadShort

type = br.ReadShort

format = br.ReadShort

flags = br.ReadShort //if second set of bytes 01 then external, if 00 then internal

reg\_point\_x = br.ReadShort

reg\_point\_y = br.ReadShort

num\_mipmaps = br.ReadShort

pixel\_offset = br.ReadShort

offset = br.ReadInt32

size = br.ReadInt32

unknown8 = br.ReadInt32

unknown9 = br.ReadInt32

unknown10 = br.ReadInt32

unknown11 = br.ReadInt32

End Sub

### **bitm\_image\_info.write:**

Sub write(byref bw as binaryStream)

bw.LittleEndian = true

bw.Position = position

bw.Write(class1)

bw.WriteShort(width)

bw.WriteShort(height)

bw.WriteShort(depth)

bw.WriteShort(type)

bw.WriteShort(format)

bw.WriteShort(flags)

bw.WriteShort(reg\_point\_x)

bw.WriteShort(reg\_point\_y)

bw.WriteShort(num\_mipmaps)

bw.WriteShort(pixel\_offset)

bw.WriteInt32(offset)

bw.WriteInt32(size)

bw.WriteInt32(unknown8)

bw.WriteInt32(unknown9)

bw.WriteInt32(unknown10)

bw.WriteInt32(unknown11)

End Sub

class1 As string

depth As short

flags As short

format As short

height As short

num\_mipmaps As short

offset As Integer

pixel\_offset As short

position As Integer

reg\_point\_x As short

reg\_point\_y As short

size As Integer

type As short

unknown10 As Integer

unknown11 As Integer

unknown8 As Integer

unknown9 As Integer

width As short

## **bitm\_image\_info Note: enums**

enums

//formats

```
#define BITM_FORMAT_A8      0x00
#define BITM_FORMAT_Y8      0x01
#define BITM_FORMAT_AY8     0x02
#define BITM_FORMAT_A8Y8    0x03
#define BITM_FORMAT_R5G6B5   0x06
#define BITM_FORMAT_A1R5G5B5 0x08
#define BITM_FORMAT_A4R4G4B4 0x09
#define BITM_FORMAT_X8R8G8B8 0x0A
#define BITM_FORMAT_A8R8G8B8 0x0B
#define BITM_FORMAT_DXT1     0x0E
#define BITM_FORMAT_DXT2AND3 0x0F
#define BITM_FORMAT_DXT4AND5 0x10
#define BITM_FORMAT_P8       0x11
```

// Types

```
#define BITM_TYPE_2D      0x00
#define BITM_TYPE_3D      0x01
#define BITM_TYPE_CUBEMAP 0x02
```

End Class

## **Class bitm\_image**

### **bitm\_image.add\_layers:**

```
Sub add_layers(p() as picture, a() as picture)
    redim layers(-1)
    for i as integer = 0 to UBound(p)
        dim temp_layer as new bitm_image(p(i), a(i))
        layers.Append temp_layer
    next
End Sub
```

### **bitm\_image.add\_layers:**

```
Sub add_layers(p() as picture)
    redim layers(-1)
    for i as integer = 0 to UBound(p)
        dim temp_layer as new bitm_image(p(i))
        layers.Append temp_layer
    next
End Sub
```

### **bitm\_image.Constructor:**

```
Sub Constructor(in_image as picture)
    image = in_image
    composite = in_image
    mask_flag = false
End Sub
```

### **bitm\_image.Constructor:**

```
Sub Constructor(in_image as picture, in_mask as picture)
    image = in_image
    mask = in_mask
    composite = in_image
    composite.Mask = in_mask
    mask_flag = true
End Sub
composite As picture
```

image As picture

layers(-1) As bitm\_image

mask As picture

mask\_flag As boolean

mip\_maps(-1) As bitm\_image

End Class

Class cube\_map

### **cube\_map.Constructor:**

Sub Constructor(input() as bitm\_image)

face = input

End Sub

face(-1) As bitm\_image

End Class

## **Module bitm\_import**

### **bitm\_import.color\_to\_uint16:**

Private Function color\_to\_uint16(input as color) As uint16

dim output as UInt16 = 0

output = \_

Bitwise.ShiftLeft(BitAnd( (input.Red/8), &h1F ), 11) + \_

Bitwise.ShiftLeft(BitAnd( (input.Green/4), &h3F ), 5) + \_

Bitwise.ShiftLeft(BitAnd( (input.Blue/8), &h1F), 0)

Return output

End Function

### **bitm\_import.covariance:**

Private Function covariance(data1() as double, data2() as double) As double

dim mean1 as double = mean(data1)

dim mean2 as double = mean(data2)

dim length as double = data1.ubound

if mean1 = 0 or mean2 = 0 then

Return 0

end

if data1.Ubound <> data2.Ubound or data1.Ubound <= 0 then

Return 0

end

dim sum as double = 0.0

for i as Integer = 0 to data1.Ubound

dim temp1 as double = data1(i) - mean1

dim temp2 as double = data2(i) - mean2

dim temp3 as double = temp2\*temp1

sum = sum + temp3

next

Return sum/length //ubound = n-1 due to zero indexing

End Function

### **bitm\_import.create\_A1R5G5B5:**

Private Function create\_A1R5G5B5(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

data.LittleEndian = true



```

dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset
for y as integer = 0 to input_image.height-1
    for x as integer = 0 to input_mask.width-1
        dim temp_color as color = input_image.RGBSurface.pixel(X,Y)
        dim temp_alpha as integer = min(input_mask.RGBSurface.pixel(X,Y).red, _
input_mask.RGBSurface.pixel(X,Y).green, input_mask.RGBSurface.pixel(X,Y).blue)
        dim value as UInt16 = 0
        value = _
        Bitwise.ShiftLeft(BitAnd( (temp_alpha), &h1), 15) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Red/8), &h1F ), 10) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Green/8), &h1F ), 5) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Blue/8), &h1F), 0)
        bw.Writeuint16(value)
    next
next
offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_A4R4G4B4:**

Private Function create\_A4R4G4B4(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

```

data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset
for y as integer = 0 to input_image.height-1
    for x as integer = 0 to input_mask.width-1
        dim temp_color as color = input_image.RGBSurface.pixel(X,Y)
        dim temp_alpha as integer = max(input_mask.RGBSurface.pixel(X,Y).red, _
input_mask.RGBSurface.pixel(X,Y).green, input_mask.RGBSurface.pixel(X,Y).blue)
        dim value as UInt16 = 0
        value = _
        Bitwise.ShiftLeft(BitAnd( (temp_alpha/17), &h0F), 12) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Red/17), &h0F ), 8) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Green/17), &h0F ), 4) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Blue/17), &h0F), 0)
        bw.Writeuint16(value)
    next
next
offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_A8:**

Private Function create\_A8(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock

```

data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset

```

```

for y as integer = 0 to input.height-1
    for x as integer = 0 to input.width-1
        dim temp_color as color = input.RGBSurface.pixel(X,Y)
        dim value as uint8 = max(temp_color.red, temp_color.green, temp_color.blue)
        bw.Writeuint8(value)
    next
next
offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_A8R8G8B8:**

Private Function create\_A8R8G8B8(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

```

data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset
for y as integer = 0 to input_image.height-1
    for x as integer = 0 to input_mask.width-1
        dim temp_color as color = input_image.RGBSurface.pixel(X,Y)
        dim temp_alpha as integer = max(input_mask.RGBSurface.pixel(X,Y).red, _
input_mask.RGBSurface.pixel(X,Y).green, input_mask.RGBSurface.pixel(X,Y).blue)
        dim value as UInt32 = 0
        value = _
        Bitwise.ShiftLeft(BitAnd( (temp_alpha), &hFF), 24) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Red), &hFF ), 16) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Green), &hFF ), 8) + _
        Bitwise.ShiftLeft(BitAnd( (temp_color.Blue), &hFF), 0)
        bw.Writeuint32(value)
    next
next
offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_A8Y8:**

Private Function create\_A8Y8(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

```

data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset
for y as integer = 0 to input_image.height-1
    for x as integer = 0 to input_image.width-1
        dim temp_color2 as color = input_mask.RGBSurface.pixel(X,Y)
        dim value as uint8 = max(temp_color2.red, temp_color2.green, temp_color2.blue)
        bw.Writeuint8(value)
    next
next
offset = bw.Position
bw.Close

```

```
    return data
End Function
```

### **bitm\_import.create\_AY8:**

Private Function create\_AY8(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

```
    data.LittleEndian = true
    dim bw as new BinaryStream(data)
    bw.LittleEndian = true
    bw.Position = offset
    for y as integer = 0 to input_image.height-1
        for x as integer = 0 to input_image.width-1
            dim temp_color as color = input_image.RGBSurface.pixel(X,Y)
            dim temp_color2 as color = input_mask.RGBSurface.pixel(X,Y)
            dim value as uint8 = max(temp_color.red, temp_color.green, temp_color.blue, _
            temp_color2.red, temp_color2.green, temp_color2.blue)
            bw.Writeuint8(value)
        next
    next
    offset = bw.Position
    bw.Close
    return data
End Function
```

### **bitm\_import.create\_bitmap:**

Function create\_bitmap(info as bitm\_image\_info, target as picture) As memoryBlock

```
    dim data as new MemoryBlock(0)
    dim offset as integer = 0

    select case info.format

    case &h0 //A8

        dim current_max_dim as integer = max(info.width, info.height)
        for i as integer = 0 to info.num_mipmaps
            dim temp_pic as Picture = target.scaleit_max(current_max_dim)
            data = create_A8(data, offset, temp_pic)
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        next

    case &h1 //Y8

        dim current_max_dim as integer = max(info.width, info.height)
        for i as integer = 0 to info.num_mipmaps
            dim temp_pic as Picture = target.scaleit_max(current_max_dim)
            data = create_Y8(data, offset, temp_pic)
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        next

    case &h2 //AY8

        //should not exist

    case &h3 //A8Y8
```

```

//should not exist

case &h6 //R5G6B5

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        dim temp_pic as Picture = target.scaleit_max(current_max_dim)
        data = create_R5G6B5(data, offset, temp_pic)
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

case &h8 //A1R5G5B5

    //should not exist

case &h9 //A4R4G4B4

    //should not exist

case &hA //X8R8G8B8

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        dim temp_pic as Picture = target.scaleit_max(current_max_dim)
        data = create_X8R8G8B8(data, offset, temp_pic)
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

case &hB //A8R8G8B8

    //should not exist

case &hE //DXT1

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        dim temp_pic as Picture = target.scaleit_max(current_max_dim)
        data = create_DXT1(data, offset, temp_pic)
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

case &hF //DXT2and3

    //should not exist

case &h10 //DXT4and5

    //should not exist

case &h11 //P8

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps

```

```

        dim temp_pic as Picture = target.scaleit_max(current_max_dim)
        data = create_P8(data, offset, temp_pic)
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

end select

if offset <> info.size then
    break
end

Return data
End Function

```

### **bitm\_import.create\_bitmap:**

Function create\_bitmap(info as bitm\_image\_info, target\_image as picture, target\_mask as picture) As memoryBlock

```

    dim data as new MemoryBlock(0)
    dim offset as integer = 0

    select case info.format

    case &h0 //A8

        //should not exist

    case &h1 //Y8

        //should not exist

    case &h2 //AY8

        dim current_max_dim as integer = max(info.width, info.height)
        for i as integer = 0 to info.num_mipmaps
            dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
            dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
            if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
                break
            end
            data = create_AY8(data, offset, temp_image, temp_mask)
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        next

    case &h3 //A8Y8

        dim current_max_dim as integer = max(info.width, info.height)
        for i as integer = 0 to info.num_mipmaps
            dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
            dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
            if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
                break
            end
            data = create_A8Y8(data, offset, temp_image, temp_mask)
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        next
    end select

```

case &h6 //R5G6B5

//should not exist

case &h8 //A1R5G5B5

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
        break
    end
    data = create_A1R5G5B5(data, offset, temp_image, temp_mask)
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &h9 //A4R4G4B4

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
        break
    end
    data = create_A4R4G4B4(data, offset, temp_image, temp_mask)
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &hA //X8R8G8B8

//should not exist

case &hB //A8R8G8B8

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
        break
    end
    data = create_A8R8G8B8(data, offset, temp_image, temp_mask)
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &hE //DXT1

//should not exist

case &hF //DXT2and3

```

dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
        break
    end
    data = create_DXT2_3(data, offset, temp_image, temp_mask)
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next

```

case &h10 //DXT4and5

```

dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    dim temp_image as Picture = target_image.scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_mask.scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
        break
    end
    data = create_DXT4_5(data, offset, temp_image, temp_mask)
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next

```

case &h11 //P8

//should not exist

end select

```

if offset <> info.size then
    break
end

```

Return data

End Function

### **bitm\_import.create\_bitmaps:**

Function create\_bitmaps(info as bitm\_image\_info, targets() as picture) As memoryBlock

```

dim data as new MemoryBlock(0)

```

```

dim offset as integer = 0

```

```

select case info.format

```

case &h0 //A8

```

dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    for j as integer = 0 to UBound(targets)
        dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
        data = create_A8(data, offset, temp_pic)
    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next

```

case &h1 //Y8

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(targets)
    dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
    data = create_Y8(data, offset, temp_pic)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &h2 //AY8

//should not exist

case &h3 //A8Y8

//should not exist

case &h6 //R5G6B5

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(targets)
    dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
    data = create_R5G6B5(data, offset, temp_pic)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &h8 //A1R5G5B5

//should not exist

case &h9 //A4R4G4B4

//should not exist

case &hA //X8R8G8B8

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(targets)
    dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
    data = create_X8R8G8B8(data, offset, temp_pic)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

case &hB //A8R8G8B8

//should not exist



```
case &hE //DXT1
```

```
    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to UBound(targets)
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
            data = create_DXT1(data, offset, temp_pic)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next
```

```
case &hF //DXT2and3
```

```
    //should not exist
```

```
case &h10 //DXT4and5
```

```
    //should not exist
```

```
case &h11 //P8
```

```
    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to UBound(targets)
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
            data = create_P8(data, offset, temp_pic)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next
```

```
end select
```

```
if offset <> info.size then
```

```
    break
```

```
end
```

```
Return data
```

```
End Function
```

### **bitm\_import.create\_bitmaps:**

Function create\_bitmaps(info as bitm\_image\_info, target\_images() as picture, target\_masks() as picture) As memoryBlock

```
    dim data as new MemoryBlock(0)
```

```
    dim offset as integer = 0
```

```
    if target_images.ubound <> target_masks.ubound then Return data
```

```
    select case info.format
```

```
    case &h0 //A8
```

```
        //should not exist
```

```
    case &h1 //Y8
```

```
//should not exist
```

```
case &h2 //AY8
```

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(Target_images)
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_AY8(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

```
case &h3 //A8Y8
```

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(Target_images)
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_A8Y8(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

```
case &h6 //R5G6B5
```

```
//should not exist
```

```
case &h8 //A1R5G5B5
```

```
dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(Target_images)
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_A1R5G5B5(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next
```

```
case &h9 //A4R4G4B4
```

```

dim current_max_dim as integer = max(info.width, info.height)
for i as integer = 0 to info.num_mipmaps
    for j as integer = 0 to UBound(Target_images)
        dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
        dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
        if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
            break
        end
        data = create_A4R4G4B4(data, offset, temp_image, temp_mask)
    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next

case &hA //X8R8G8B8

    //should not exist

case &hB //A8R8G8B8

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to UBound(Target_images)
            dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
            dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
            if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
                break
            end
            data = create_A8R8G8B8(data, offset, temp_image, temp_mask)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

case &hE //DXT1

    //should not exist

case &hF //DXT2and3

    dim current_max_dim as integer = max(info.width, info.height)
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to UBound(Target_images)
            dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
            dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
            if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
                break
            end
            data = create_DXT2_3(data, offset, temp_image, temp_mask)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    next

case &h10 //DXT4and5

    dim current_max_dim as integer = max(info.width, info.height)

```

```

for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to UBound(Target_images)
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_DXT4_5(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
next

case &h11 //P8

  //should not exist

end select

if offset <> info.size then
  break
end

Return data
End Function

```

### **bitm\_import.create\_DXT1:**

Private Function create\_DXT1(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock

```

data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset

//dim output as Picture = new Picture(input.width, input.height, 32)

if min(input.height, input.width) < 4 then
  if min(input.height, input.width) = 1 then
    bw.writeuint16(color_to_uint16(input.RGBSurface.pixel(0,0)))
    bw.writeuint16(color_to_uint16(input.RGBSurface.pixel(0,0)))
  else
    bw.writeuint16(color_to_uint16(input.RGBSurface.pixel(0,0)))
    bw.writeuint16(color_to_uint16(input.RGBSurface.pixel(input.width-1,input.height-1)))
  end
  bw.writeuint32(0)
else
  for y as integer = 0 to input.height/4 - 1
    for x as integer = 0 to input.width/4 - 1

      dim c(-1) as color
      for j as integer = 0 to 3
        for i as integer = 0 to 3
          c.append input.RGBSurface.pixel(i+(x*4), j+(y*4))
        next
      next
    next
  next

```

```
dim temp_val as UInt64 = linearize_DXT1(c)
bw.WriteUInt64(temp_val)
```

```
//get the linearized image
'dim temp() as color = linearize(c)
'dim index as integer = 0
'for j as integer = 0 to 3
'for i as integer = 0 to 3
'output.RGBSurface.pixel(i+x*4, j+y*4) = temp(index)
'index = index+1
'next
'next
```

```
next
next
end
```

```
offset = bw.Position
bw.Close
return data
End Function
```

### **bitm\_import.create\_DXT2\_3:**

Private Function create\_DXT2\_3(data as memoryBlock, byref offset as integer, input\_image as picture, input\_mask as picture) As memoryBlock

```
data.LittleEndian = true
dim bw as new BinaryStream(data)
bw.LittleEndian = true
bw.Position = offset
```

```
if min(input_image.height, input_image.width) < 4 then
```

```
    //alpha portion
    dim temp_alpha_val as integer = max(_
    input_mask.RGBSurface.pixel(0,0).red, _
    input_mask.RGBSurface.pixel(0,0).green, _
    input_mask.RGBSurface.pixel(0,0).blue)
    temp_alpha_val = temp_alpha_val/17
    dim bit as integer = BitAnd(temp_alpha_val, &hF)
    dim output as uint16 = _
    Bitwise.ShiftLeft(bit, 12) + _
    Bitwise.ShiftLeft(bit, 8) + _
    Bitwise.ShiftLeft(bit, 4) + _
    Bitwise.ShiftLeft(bit, 0)
    bw.writeuint16( output )
    bw.writeuint16( output )
    bw.writeuint16( output )
    bw.writeuint16( output )
```

```
    //color portion
    if min(input_image.height, input_image.width) = 1 then
        bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
        bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
    else
```

```

        bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
        bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(input_image.width-1,input_image.height-1)))
    end
    bw.writeuint32(0)
else
    for y as integer = 0 to input_image.height/4 - 1
        for x as integer = 0 to input_image.width/4 - 1

            //alpha portion
            for j as integer = 0 to 3
                dim temp_int as uint16 = 0
                for i as integer = 0 to 3
                    dim temp_acolor as color = input_mask.RGBSurface.pixel(i+(x*4),j+(y*4))
                    dim temp_alpha as integer = min(temp_acolor.red, temp_acolor.green, temp_acolor.blue)
                    temp_alpha = temp_alpha/16
                    temp_alpha = BitAnd(temp_alpha, &hF)
                    temp_int = temp_int + Bitwise.ShiftLeft(temp_alpha, i*4)
                next
                bw.writeuint16(temp_int)
            next

            //color portion
            dim c(-1) as color
            for j as integer = 0 to 3
                for i as integer = 0 to 3
                    c.append input_image.RGBSurface.pixel(i+(x*4), j+(y*4))
                next
            next

            dim temp_val as UInt64 = linearize_DXT1(c)
            bw.WriteUInt64(temp_val)

        next
    next
end

offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_DXT4\_5:**

```

Private Function create_DXT4_5(data as memoryBlock, byref offset as integer, input_image as picture, input_mask as picture) As memoryBlock
    data.LittleEndian = true
    dim bw as new BinaryStream(data)
    bw.LittleEndian = true
    bw.Position = offset

    if min(input_image.height, input_image.width) < 4 then
        //alpha portion
        dim max_alpha as uint8 = max(_
        input_mask.RGBSurface.pixel(0,0).red, _

```

```

input_mask.RGBSurface.pixel(0,0).green, _
input_mask.RGBSurface.pixel(0,0).blue )
dim min_alpha as uint8 = max(_
input_mask.RGBSurface.pixel(input_mask.width,input_mask.height).red, _
input_mask.RGBSurface.pixel(input_mask.width,input_mask.height).green, _
input_mask.RGBSurface.pixel(input_mask.width,input_mask.height).blue )
if min_alpha > max_alpha then
    dim temp_alpha_val as uint8 = max_alpha
    max_alpha = min_alpha
    min_alpha = temp_alpha_val
end
while min_alpha = max_alpha
    min_alpha = min_alpha - 1
wend
bw.writeuint8(max_alpha)
bw.writeuint8(min_alpha)
bw.writeuint16(0)
bw.writeuint32(0)

//color portion
if min(input_image.height, input_image.width) = 1 then
    bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
    bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
else
    bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(0,0)))
    bw.writeuint16(color_to_uint16(input_image.RGBSurface.pixel(input_image.width-1,input_image.height-1)))
end
bw.writeuint32(0)
else
    for y as integer = 0 to input_image.height/4 - 1
        for x as integer = 0 to input_image.width/4 - 1

            //alpha portion
            dim alpha_set(-1) as integer
            dim max_alpha as uint8 = 0
            dim min_alpha as uint8 = 0
            for j as integer = 0 to 3
                for i as integer = 0 to 3
                    dim temp_acolor as color = input_mask.RGBSurface.pixel(i+(x*4),j+(y*4))
                    dim temp_alpha as uint8 = min(temp_acolor.red, temp_acolor.green, temp_acolor.blue)
                    if i = 0 and j = 0 then
                        max_alpha = temp_alpha
                        min_alpha = temp_alpha
                    end
                    if max_alpha < temp_alpha then
                        max_alpha = temp_alpha
                    end
                    if min_alpha > temp_alpha then
                        min_alpha = temp_alpha
                    end
                    alpha_set.append temp_alpha
                next
            next
        next
    next

```

```

dim a0, a1, a2, a3, a4, a5, a6, a7 as uint8
if max_alpha < min_alpha then
    dim temp as integer = max_alpha
    max_alpha = min_alpha
    min_alpha = temp
end
while max_alpha = min_alpha
    min_alpha = min_alpha - 1
wend
a0 = max_alpha
a1 = min_alpha
a2 = (6*a0 + 1*a1)/7
a3 = (5*a0 + 2*a1)/7
a4 = (4*a0 + 3*a1)/7
a5 = (3*a0 + 4*a1)/7
a6 = (2*a0 + 5*a1)/7
a7 = (1*a0 + 6*a1)/7
//lookup table: a_value = 0-7 , 3bits AND 0x7
dim alpha_lookup(-1) as uint8
for i as integer = 0 to UBound(alpha_set)
    select case min(
        abs(alpha_set(i) - a0), _
        abs(alpha_set(i) - a1), _
        abs(alpha_set(i) - a2), _
        abs(alpha_set(i) - a3), _
        abs(alpha_set(i) - a4), _
        abs(alpha_set(i) - a5), _
        abs(alpha_set(i) - a6), _
        abs(alpha_set(i) - a7) )

        case abs(alpha_set(i) - a0)
            alpha_lookup.append 0
        case abs(alpha_set(i) - a1)
            alpha_lookup.append 1
        case abs(alpha_set(i) - a2)
            alpha_lookup.append 2
        case abs(alpha_set(i) - a3)
            alpha_lookup.append 3
        case abs(alpha_set(i) - a4)
            alpha_lookup.append 4
        case abs(alpha_set(i) - a5)
            alpha_lookup.append 5
        case abs(alpha_set(i) - a6)
            alpha_lookup.append 6
        case abs(alpha_set(i) - a7)
            alpha_lookup.append 7
    end select
next
bw.writeuint8(a0)
bw.writeuint8(a1)

dim output_str as string = ""
for i as integer = 0 to UBound(alpha_lookup)
    dim temp_str as string = bin(bitand(alpha_lookup(i), &h7))

```



```

        while len(temp_str) < 3
            temp_str = "0" + temp_str
        wend
        output_str = output_str + temp_str
    next
    dim output_data as new MemoryBlock(64)
    output_data.LittleEndian = true
    output_data.uint64value(0) = val("&b" + output_str)
    dim br as new BinaryStream(output_data)
    br.LittleEndian = true
    br.position = 4
    bw.writeuint16(br.readuint16)
    bw.writeuint32(br.readuint32)
    br.close

    //color portion
    dim c(-1) as color
    for j as integer = 0 to 3
        for i as integer = 0 to 3
            c.append input_image.RGBSurface.pixel(i+(x*4), j+(y*4))
        next
    next

    dim temp_val as UInt64 = linearize_DXT1(c)
    bw.WriteUInt64(temp_val)

next
next
end

offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_P8:**

```

Private Function create_P8(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock
    data.LittleEndian = true
    dim bw as new BinaryStream(data)
    bw.LittleEndian = true
    bw.Position = offset
    for y as integer = 0 to input.height-1
        for x as integer = 0 to input.width-1
            dim temp_color as color = input.RGBSurface.pixel(X,Y)
            dim value as uint8 = max(temp_color.red, temp_color.green, temp_color.blue)
            bw.Writeuint8(value)
        next
    next
    offset = bw.Position
    bw.Close
    return data
End Function

```

### **bitm\_import.create\_R5G6B5:**

```

Private Function create_R5G6B5(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock
    data.LittleEndian = true
    dim bw as new BinaryStream(data)
    bw.LittleEndian = true
    bw.Position = offset
    for y as integer = 0 to input.height-1
        for x as integer = 0 to input.width-1
            dim temp_color as color = input.RGBSurface.pixel(X,Y)
            dim value as uint16 = color_to_uint16(temp_color)
            bw.Writeuint16(value)
        next
    next
    offset = bw.Position
    bw.Close
    return data
End Function

```

### **bitm\_import.create\_volume\_bitmaps:**

```

Function create_volume_bitmaps(info as bitm_image_info, targets() as picture) As memoryBlock
    dim data as new MemoryBlock(0)
    dim offset as integer = 0

    select case info.format

    case &h0 //A8

        dim current_max_dim as integer = max(info.width, info.height)
        dim current_layer_max as integer = info.depth - 1
        if current_layer_max > ubound(targets) then break
        for i as integer = 0 to info.num_mipmaps
            for j as integer = 0 to current_layer_max
                dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
                data = create_A8(data, offset, temp_pic)
            next
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
            current_layer_max = current_layer_max/2
        next

    case &h1 //Y8

        dim current_max_dim as integer = max(info.width, info.height)
        dim current_layer_max as integer = info.depth - 1
        if current_layer_max > ubound(targets) then break
        for i as integer = 0 to info.num_mipmaps
            for j as integer = 0 to current_layer_max
                dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
                data = create_Y8(data, offset, temp_pic)
            next
            current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
            current_layer_max = current_layer_max/2
        next

    case &h2 //AY8

```

```

//should not exist

case &h3 //A8Y8

    //should not exist

case &h6 //R5G6B5

    dim current_max_dim as integer = max(info.width, info.height)
    dim current_layer_max as integer = info.depth - 1
    if current_layer_max > ubound(targets) then break
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to current_layer_max
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
            data = create_R5G6B5(data, offset, temp_pic)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        current_layer_max = current_layer_max/2
    next

case &h8 //A1R5G5B5

    //should not exist

case &h9 //A4R4G4B4

    //should not exist

case &hA //X8R8G8B8

    dim current_max_dim as integer = max(info.width, info.height)
    dim current_layer_max as integer = info.depth - 1
    if current_layer_max > ubound(targets) then break
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to current_layer_max
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
            data = create_X8R8G8B8(data, offset, temp_pic)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        current_layer_max = current_layer_max/2
    next

case &hB //A8R8G8B8

    //should not exist

case &hE //DXT1

    dim current_max_dim as integer = max(info.width, info.height)
    dim current_layer_max as integer = info.depth - 1
    if current_layer_max > ubound(targets) then break
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to current_layer_max
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)

```

```

        data = create_DXT1(data, offset, temp_pic)
    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    current_layer_max = current_layer_max/2
next

case &hF //DXT2and3

    //should not exist

case &h10 //DXT4and5

    //should not exist

case &h11 //P8

    dim current_max_dim as integer = max(info.width, info.height)
    dim current_layer_max as integer = info.depth - 1
    if current_layer_max > ubound(targets) then break
    for i as integer = 0 to info.num_mipmaps
        for j as integer = 0 to current_layer_max
            dim temp_pic as Picture = targets(j).scaleit_max(current_max_dim)
            data = create_P8(data, offset, temp_pic)
        next
        current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
        current_layer_max = current_layer_max/2
    next

end select

if offset <> info.size then
    break
end

Return data
End Function

```

### **bitm\_import.create\_volume\_bitmaps:**

Function create\_volume\_bitmaps(info as bitm\_image\_info, target\_images() as picture, target\_masks() as picture) As memoryBlock

```

    dim data as new MemoryBlock(0)
    dim offset as integer = 0

    if target_images.ubound <> target_masks.ubound then Return data

    select case info.format

    case &h0 //A8

        //should not exist

    case &h1 //Y8

        //should not exist

```

case &h2 //AY8

```
dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to current_layer_max
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_AY8(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
  current_layer_max = current_layer_max/2
next
```

case &h3 //A8Y8

```
dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to current_layer_max
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_A8Y8(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
  current_layer_max = current_layer_max/2
next
```

case &h6 //R5G6B5

//should not exist

case &h8 //A1R5G5B5

```
dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to current_layer_max
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_A1R5G5B5(data, offset, temp_image, temp_mask)
```

```

    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    current_layer_max = current_layer_max/2
next

case &h9 //A4R4G4B4

dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
    for j as integer = 0 to current_layer_max
        dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
        dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
        if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
            break
        end
        data = create_A4R4G4B4(data, offset, temp_image, temp_mask)
    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    current_layer_max = current_layer_max/2
next

case &hA //X8R8G8B8

    //should not exist

case &hB //A8R8G8B8

dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
    for j as integer = 0 to current_layer_max
        dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
        dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
        if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
            break
        end
        data = create_A8R8G8B8(data, offset, temp_image, temp_mask)
    next
    current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
    current_layer_max = current_layer_max/2
next

case &hE //DXT1

    //should not exist

case &hF //DXT2and3

dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break

```

```

for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to current_layer_max
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_DXT2_3(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
  current_layer_max = current_layer_max/2
next

```

case &h10 //DXT4and5

```

dim current_max_dim as integer = max(info.width, info.height)
dim current_layer_max as integer = info.depth - 1
if current_layer_max > ubound(target_images) then break
for i as integer = 0 to info.num_mipmaps
  for j as integer = 0 to current_layer_max
    dim temp_image as Picture = target_images(j).scaleit_max(current_max_dim)
    dim temp_mask as Picture = target_masks(j).scaleit_max(current_max_dim)
    if temp_image.Width <> temp_mask.Width or temp_image.Height <> temp_mask.Height then
      break
    end
    data = create_DXT4_5(data, offset, temp_image, temp_mask)
  next
  current_max_dim = max(current_max_dim/2, 1) //make sure that it doesn't try a zero width/height image
  current_layer_max = current_layer_max/2
next

```

case &h11 //P8

//should not exist

end select

```

if offset <> info.size then
  break
end

```

Return data

End Function

### **bitm\_import.create\_X8R8G8B8:**

```

Private Function create_X8R8G8B8(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock
  data.LittleEndian = true
  dim bw as new BinaryStream(data)
  bw.LittleEndian = true
  bw.Position = offset
  for y as integer = 0 to input.height-1
    for x as integer = 0 to input.width-1
      dim temp_color as color = input.RGBSurface.pixel(X,Y)
      dim value as uint32 = _
    next x
  next y

```

```

        Bitwise.ShiftLeft(BitAnd( temp_color.Red, &hFF ), 16) + _
        Bitwise.ShiftLeft(BitAnd( temp_color.Green, &hFF ), 8) + _
        Bitwise.ShiftLeft(BitAnd( temp_color.Blue, &hFF), 0)
    bw.Writeuint32(value)
next
next
offset = bw.Position
bw.Close
return data
End Function

```

### **bitm\_import.create\_Y8:**

```

Private Function create_Y8(data as memoryBlock, byref offset as integer, input as picture) As memoryBlock
    data.LittleEndian = true
    dim bw as new BinaryStream(data)
    bw.LittleEndian = true
    bw.Position = offset
    for y as integer = 0 to input.height-1
        for x as integer = 0 to input.width-1
            dim temp_color as color = input.RGBSurface.pixel(X,Y)
            dim value as uint8 = max(temp_color.red, temp_color.green, temp_color.blue)
            bw.Writeuint8(value)
        next
    next
    offset = bw.Position
    bw.Close
    return data
End Function

```

### **bitm\_import.dot\_product:**

```

Private Function dot_product(vector1() as double, vector2() as double) As double
    if Vector1.Ubound <> Vector2.Ubound then Return 0
    dim output as double = 0
    for i as Integer = 0 to Vector1.Ubound
        output = output + Vector1(i)*Vector2(i)
    next
    Return output
End Function

```

### **bitm\_import.linearize:**

```

Private Function linearize(data() as color) As color()
    if data.ubound <> 15 then
        break
        //return the zero vector if the data is incorrect
    dim temp() as color
    for i as integer = 1 to 16
        temp.append rgb(255,255,255)
    next
    Return temp
end

//create the basic 3D arrays
dim r_ar() as double

```



```

dim g_ar() as double
dim b_ar() as double
for i as integer = 0 to UBound(data)
    r_ar.Append data(i).Red
    g_ar.Append data(i).Green
    b_ar.Append data(i).Blue
next

//remove the mean values along all dimensions
dim r_mean as double = mean(r_ar)
dim g_mean as double = mean(g_ar)
dim b_mean as double = mean(b_ar)
for i as integer = 0 to UBound(data)
    r_ar(i) = r_ar(i) - r_mean
    g_ar(i) = g_ar(i) - g_mean
    b_ar(i) = b_ar(i) - b_mean
next

//create the covariance matrix
// x by y
// x tall
// y wide
// matrix.element(i,j)
// i is the row
// j is the column
dim cov_rr as double = covariance(r_ar, r_ar)
dim cov_gr as double = covariance(g_ar, r_ar)
dim cov_br as double = covariance(b_ar, r_ar)
dim cov_rg as double = covariance(r_ar, g_ar)
dim cov_gg as double = covariance(g_ar, g_ar)
dim cov_bg as double = covariance(b_ar, g_ar)
dim cov_rb as double = covariance(r_ar, b_ar)
dim cov_gb as double = covariance(g_ar, b_ar)
dim cov_bb as double = covariance(b_ar, b_ar)
//use the complex method instead
'dim mat_str as string = _
'str(cov_rr) + "," + str(cov_rg) + "," + str(cov_rb) + ";" + _
'str(cov_gr) + "," + str(cov_gg) + "," + str(cov_gb) + ";" + _
'str(cov_br) + "," + str(cov_bg) + "," + str(cov_bb)
'dim cov_matrix as new Matrix(3,3, mat_str)
'dim eigen_vector_matrix as new Matrix(3,3)
'dim eigen_value_matrix as Matrix = Diagonalize(cov_matrix, eigen_vector_matrix, 1e-10)
'dim max_index as integer = 0
'if eigen_value_matrix.Element(1,1) > eigen_value_matrix.Element(0,0) then max_index = 1
'if eigen_value_matrix.Element(2,2) > eigen_value_matrix.Element(max_index, max_index) then max_index = 2
'dim r_coord as double = eigen_vector_matrix.Element(0, max_index)
'dim g_coord as double = eigen_vector_matrix.Element(1, max_index)
'dim b_coord as double = eigen_vector_matrix.Element(2, max_index)
if cov_gr <> cov_rg OR cov_br <> cov_rb OR cov_gb <> cov_bg then
    break
end

dim mat_str as string = _
str(cov_rr) + ",0 " + str(cov_rg) + ",0 " + str(cov_rb) + ",0 ;" + _

```

```

str(cov_rg) + ",0 " + str(cov_gg) + ",0 " + str(cov_gb) + ",0;" + _
str(cov_rb) + ",0 " + str(cov_gb) + ",0 " + str(cov_bb) + ",0 "
dim cov_matrix as new ComplexMatrix(3,3, mat_str)
dim eigen_vector_matrix as new ComplexMatrix(3,3)
dim eigen_value_matrix as ComplexMatrix = Diagonalize(cov_matrix, eigen_vector_matrix, 1e-10)
dim max_index as integer = 0
//all eigen_values should be real because the matrix is symetric
if eigen_value_matrix.Element(1,1) > eigen_value_matrix.Element(0,0) then max_index = 1
if eigen_value_matrix.Element(2,2) > eigen_value_matrix.Element(max_index, max_index) then max_index = 2
dim temp_str as string = eigen_value_matrix.Str
dim temp_str2 as string = eigen_vector_matrix.str
dim r_coord as double = eigen_vector_matrix.Element(0, max_index).real
dim g_coord as double = eigen_vector_matrix.Element(1, max_index).real
dim b_coord as double = eigen_vector_matrix.Element(2, max_index).real

//normalize the coordinates
dim length_vector as double = sqrt(r_coord^2 + g_coord^2 + b_coord^2)
r_coord = r_coord/length_vector
g_coord = g_coord/length_vector
b_coord = b_coord/length_vector

//find the max and min value along the scale, then determine the 2/9th and 8/9th value along that line
dim dist_ar() as double
dim principle_axis() as double = array(r_coord, g_coord, b_coord)
for i as integer = 0 to UBound(data)
    dim temp_vector() as double = array(r_ar(i), g_ar(i), b_ar(i))
    dist_ar.Append dot_product(principle_axis, temp_vector)
    //dist_ar.Append cross_product_mag(principle_axis, temp_vector)
next
dim temp_min as double = dist_ar(0)
dim temp_max as double = dist_ar(0)
dim temp_min_index as integer = 0
dim temp_max_index as integer = 0
for i as integer = 0 to UBound(dist_ar)
    if dist_ar(i) < temp_min then
        temp_min = dist_ar(i)
        temp_min_index = i
    end
    if dist_ar(i) > temp_max then
        temp_max = dist_ar(i)
        temp_max_index = i
    end
end
next
dim temp_length as double = temp_max - temp_min
dim PCA_min as double = temp_min + (temp_length*(1.0/8.0)) //0.13
dim PCA_max as double = temp_min + (temp_length*(7.0/8.0)) //0.916
dim PCA_min_half as double = temp_min + (temp_length*(3.0/8.0)) //0.39583
dim PCA_max_half as double = temp_min + (temp_length*(5.0/8.0)) //0.65625
//0 = min, 1 = min_half, 2 = max_half, 3 = max
dim temp_output(-1) as integer
for i as integer = 0 to UBound(data)
    if abs(dist_ar(i) - PCA_min) >= abs(dist_ar(i) - PCA_max) then
        //min is further from target color than max
        if abs(dist_ar(i) - PCA_max) >= abs(dist_ar(i) - PCA_max_half) then

```

```

        //max is further than max_half
        temp_output.Append 2
    else
        //max is closer than max_half
        temp_output.Append 3
    end
else
    //max is further from target color than min
    if abs(dist_ar(i) - PCA_min) >= abs(dist_ar(i) - PCA_min_half) then
        //min is further than min_half
        temp_output.Append 1
    else
        //min is closer than min_half
        temp_output.Append 0
    end
end
end
next
//create the colors
dim r_min as double = r_mean + (PCA_min*r_coord)
dim g_min as double = g_mean + (PCA_min*g_coord)
dim b_min as double = b_mean + (PCA_min*b_coord)
dim min_color as color = RGB(r_min, g_min, b_min)
dim r_min_half as double = r_mean + (PCA_min_half*r_coord)
dim g_min_half as double = g_mean + (PCA_min_half*g_coord)
dim b_min_half as double = b_mean + (PCA_min_half*b_coord)
dim min_half_color as color = RGB(r_min_half, g_min_half, b_min_half)
dim r_max_half as double = r_mean + (PCA_max_half*r_coord)
dim g_max_half as double = g_mean + (PCA_max_half*g_coord)
dim b_max_half as double = b_mean + (PCA_max_half*b_coord)
dim max_half_color as color = RGB(r_max_half, g_max_half, b_max_half)
dim r_max as double = r_mean + (PCA_max*r_coord)
dim g_max as double = g_mean + (PCA_max*g_coord)
dim b_max as double = b_mean + (PCA_max*b_coord)
dim max_color as color = RGB(r_max, g_max, b_max)
dim output(-1) as color
for i as integer = 0 to temp_output.Ubound
    select case temp_output(i)
    case 0
        output.Append min_color
    case 1
        output.Append min_half_color
    case 2
        output.Append max_half_color
    case 3
        output.Append max_color
    end select
next
Return output
End Function

```

### **bitm\_import.linearize\_DXT1:**

```

Private Function linearize_DXT1(data() as color) As uint64
    if data.ubound <> 15 then
        break
    end if

```

```

//return the zero vector if the data is incorrect
Return 0
end

```

```

//create the basic 3D arrays
dim r_ar() as double
dim g_ar() as double
dim b_ar() as double
for i as integer = 0 to UBound(data)
    r_ar.Append data(i).Red
    g_ar.Append data(i).Green
    b_ar.Append data(i).Blue
next

```

```

//remove the mean values along all dimensions
dim r_mean as double = mean(r_ar)
dim g_mean as double = mean(g_ar)
dim b_mean as double = mean(b_ar)
for i as integer = 0 to UBound(data)
    r_ar(i) = r_ar(i) - r_mean
    g_ar(i) = g_ar(i) - g_mean
    b_ar(i) = b_ar(i) - b_mean
next

```

```

//create the covariance matrix
dim cov_rr as double = covariance(r_ar, r_ar)
dim cov_gr as double = covariance(g_ar, r_ar)
dim cov_br as double = covariance(b_ar, r_ar)
dim cov_rg as double = covariance(r_ar, g_ar)
dim cov_gg as double = covariance(g_ar, g_ar)
dim cov_bg as double = covariance(b_ar, g_ar)
dim cov_rb as double = covariance(r_ar, b_ar)
dim cov_gb as double = covariance(g_ar, b_ar)
dim cov_bb as double = covariance(b_ar, b_ar)

```

```

if cov_gr <> cov_rg OR cov_br <> cov_rb OR cov_gb <> cov_bg then
    break
end

```

```

dim mat_str as string = _
str(cov_rr) + ",0 " + str(cov_rg) + ",0 " + str(cov_rb) + ",0 ;" + _
str(cov_rg) + ",0 " + str(cov_gg) + ",0 " + str(cov_gb) + ",0 ;" + _
str(cov_rb) + ",0 " + str(cov_gb) + ",0 " + str(cov_bb) + ",0 "
dim cov_matrix as new ComplexMatrix(3,3, mat_str)
dim eigen_vector_matrix as new ComplexMatrix(3,3)
dim eigen_value_matrix as ComplexMatrix = Diagonalize(cov_matrix, eigen_vector_matrix, 1e-10)
dim max_index as integer = 0
//all eigen_values should be real because the matrix is symetric
if eigen_value_matrix.Element(1,1) > eigen_value_matrix.Element(0,0) then max_index = 1
if eigen_value_matrix.Element(2,2) > eigen_value_matrix.Element(max_index, max_index) then max_index = 2
dim temp_str as string = eigen_value_matrix.Str
dim temp_str2 as string = eigen_vector_matrix.str
dim r_coord as double = eigen_vector_matrix.Element(0, max_index).real
dim g_coord as double = eigen_vector_matrix.Element(1, max_index).real

```

```

dim b_coord as double = eigen_vector_matrix.Element(2, max_index).real

//normalize the coordinates
dim length_vector as double = sqrt(r_coord^2 + g_coord^2 + b_coord^2)
r_coord = r_coord/length_vector
g_coord = g_coord/length_vector
b_coord = b_coord/length_vector

//find the max and min value along the scale, then determine the 2/9th and 8/9th value along that line
dim dist_ar() as double
dim principle_axis() as double = array(r_coord, g_coord, b_coord)
for i as integer = 0 to UBound(data)
    dim temp_vector() as double = array(r_ar(i), g_ar(i), b_ar(i))
    dist_ar.Append dot_product(principle_axis, temp_vector)
    //dist_ar.Append cross_product_mag(principle_axis, temp_vector)
next
dim temp_min as double = dist_ar(0)
dim temp_max as double = dist_ar(0)
dim temp_min_index as integer = 0
dim temp_max_index as integer = 0
for i as integer = 0 to UBound(dist_ar)
    if dist_ar(i) < temp_min then
        temp_min = dist_ar(i)
        temp_min_index = i
    end
    if dist_ar(i) > temp_max then
        temp_max = dist_ar(i)
        temp_max_index = i
    end
next
dim temp_length as double = abs(temp_max - temp_min)
dim PCA_min as double = temp_min + (temp_length*(1.0/8.0)) //0.13
dim PCA_max as double = temp_min + (temp_length*(7.0/8.0)) //0.916
dim PCA_min_half as double = temp_min + (temp_length*(3.0/8.0)) //0.39583
dim PCA_max_half as double = temp_min + (temp_length*(5.0/8.0)) //0.65625
dim temp_output(-1) as integer
for i as integer = 0 to UBound(data)
    if abs(dist_ar(i) - PCA_min) >= abs(dist_ar(i) - PCA_max) then
        //min is further from target color than max
        if abs(dist_ar(i) - PCA_max) >= abs(dist_ar(i) - PCA_max_half) then
            //max is further than max_half
            //max_half = color3
            temp_output.Append 2
        else
            //max is closer than max_half
            //max = color1
            temp_output.Append 0
        end
    else
        //max is further from target color than min
        if abs(dist_ar(i) - PCA_min) >= abs(dist_ar(i) - PCA_min_half) then
            //min is further than min_half
            //min_half = color4
            temp_output.Append 3
        end
    end
next

```

```

    else
        //min is closer than min_half
        //min = color2
        temp_output.Append 1
    end
end
next
//create the colors
dim r_min as double = r_mean + (PCA_min*r_coord)
dim g_min as double = g_mean + (PCA_min*g_coord)
dim b_min as double = b_mean + (PCA_min*b_coord)
dim min_color as color = RGB(r_min, g_min, b_min)
dim r_max as double = r_mean + (PCA_max*r_coord)
dim g_max as double = g_mean + (PCA_max*g_coord)
dim b_max as double = b_mean + (PCA_max*b_coord)
dim max_color as color = RGB(r_max, g_max, b_max)

dim min_color_val as UInt16 = color_to_uint16(min_color)
dim max_color_val as UInt16 = color_to_uint16(max_color)

if min_color_val > max_color_val then
    //fix it such that color max is larger
    for i as integer = 0 to UBound(temp_output)
        select case temp_output(i)
            case 0
                temp_output(i) = 1
            case 1
                temp_output(i) = 0
            case 2
                temp_output(i) = 3
            case 3
                temp_output(i) = 2
        end select
    next
    dim temp_color_val as uint16= max_color_val
    max_color_val = min_color_val
    min_color_val = temp_color_val
end

dim temp_sub_val as integer = 1
while min_color_val = max_color_val
    if min_color_val = 0 then
        max_color_val = 1
    end
    min_color = RGB(r_min - temp_sub_val, g_min - temp_sub_val, b_min - temp_sub_val)
    min_color_val = color_to_uint16(min_color)
    temp_sub_val = temp_sub_val + 1
wend

dim lookup_table as uint32 = 0
for i as integer = 0 to temp_output.ubound
    lookup_table = lookup_table + _
    Bitwise.ShiftLeft( BitAnd(temp_output(i), &h3) , (i)*2 )
next

```

```

//min = color1
//min_half = color3
//max_half = color4
//max = color2
'dim temp_val as UInt64 = _
'Bitwise.ShiftLeft(color_to_uint16(min_color), 48) + _
'Bitwise.ShiftLeft(color_to_uint16(max_color), 32) + _
'BitAnd(lookup_table,&hFFFFFFFF)
'Return temp_val

dim temp_data as new MemoryBlock(0)
temp_data.littleendian = true
dim bw as new BinaryStream(temp_data)
bw.littleendian = true
bw.writeuint16(max_color_val)
bw.writeuint16(min_color_val)
bw.WriteUInt32(lookup_table)
bw.position = 0
dim temp_val as uint64 = bw.readuint64
return temp_val

'dim temp_val as UInt64 = _
'Bitwise.ShiftLeft(BitAnd(lookup_table,&hFFFFFFFF), 32) + _
'Bitwise.ShiftLeft(min_color_val, 16) + _
'Bitwise.ShiftLeft(max_color_val, 0)
'Return temp_val
End Function

```

### **bitm\_import.mean:**

```

Private Function mean(data() as double) As double
    dim sum as double = 0
    dim length as double = data.ubound
    if not (length > 0) then return 0
    for i as integer = 0 to UBound(data)
        sum = sum + data(i)
    next
    Return sum/length
End Function
End Module

```

## **Class bitmap\_class\_RW**

### **bitmap\_class\_RW.Constructor:**

```

Sub Constructor(in_bitmaps_f as folderItem, in_f as folderitem, in_offset as integer, in_magic as integer)
    bitmaps_f = in_bitmaps_f
    header = new bitm_header_RW
    magic = in_magic
    f = in_f
    offset = in_offset
End Sub

```

## **bitmap\_class\_RW.extract\_DDS:**

Function extract\_DDS(index as integer) As memoryBlock

//most of this code was borrowed from the HMT source

```
dim dds as new MemoryBlock(0)
```

```
//Surface Format Header
```

```
dim dds_header as string = "DDS "
```

```
//DDSURFACEDESC2
```

```
dim size_of_structure as integer = 124
```

```
dim flags as integer = 0
```

```
dim height as integer
```

```
dim width as integer
```

```
dim PitchOrLinearSize as integer
```

```
dim depth as integer = image_info(index).depth
```

```
dim mipmapcount as integer = 0 'total number, not just the count
```

```
dim reserved1() as integer
```

```
dim reserved2 as integer = 0
```

```
flags = flags + int32(DDSEnum.DDSD_CAPS)
```

```
flags = flags + int32(DDSEnum.DDSD_PIXELFORMAT)
```

```
flags = flags + int32(DDSEnum.DDSD_WIDTH)
```

```
flags = flags + int32(DDSEnum.DDSD_HEIGHT)
```

```
select case image_info(index).format
```

```
case &hE, &hF, &h10
```

```
    flags = flags + int32(DDSEnum.DDSD_LINEARSIZE)
```

```
case else
```

```
    flags = flags + int32(DDSEnum.DDSD_PITCH)
```

```
end
```

```
if image_info(index).type = 1 then
```

```
    flags = flags + int32(DDSEnum.DDSD_DEPTH)
```

```
end
```

```
height = image_info(index).height
```

```
width = image_info(index).width
```

```
dim RGBBitCount as Integer = 0
```

```
select case image_info(index).format
```

```
case &h6 //R5G6B5
```

```
    RGBBitCount = 16
```

```
case &h8 //A1R4G5B5
```

```
    RGBBitCount = 16
```

```
case &h9 //A4R4G4B4
```

```
    RGBBitCount = 16
```

```
case &hA //X8R8G8B8
```

```
    RGBBitCount = 32
```

```
case &hB //A8R8G8B8
```

```
    RGBBitCount = 32
```

```
end select
```

```
select case image_info(index).format
```

```
case &hE, &hF, &h10 //linear
```

```
    //size in bytes of a single main image
```

```
    if image_info(index).format = int32(BitmFormat.DXT1) then
```



```

        PitchOrLinearSize = max(image_info(index).width/4, 1) * max(image_info(index).height/4, 1) * 8
    else
        PitchOrLinearSize = max(image_info(index).width/4, 1) * max(image_info(index).height/4, 1) * 16
    end
case else //pitch
    PitchOrLinearSize = image_info(index).width * (RGBBitCount/8)
end select

if image_info(index).num_mipmaps > 0 AND image_info(index).type <> 2 then
    mipmapcount = image_info(index).num_mipmaps + 1
    flags = flags + int32(DDSEnum.DDSD_MIPMAPCOUNT)
end

redim reserved1(-1)
for i as integer = 1 to 11
    reserved1.append(0)
next

//DDPIXELFORMAT
dim size as integer = 32
dim ddpixel_flags as integer = 0
dim FourCC as string
dim ddpixel_RGBBitCount as integer
dim RBitMask as integer
dim GBitMask as integer
dim BBitMask as integer
dim RGBAlphaBitMask as integer
select case image_info(index).format
case int32(BitmFormat.DXT1)
    FourCC = "DXT1"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case int32(BitmFormat.DXT2_3)
    FourCC = "DXT3"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case int32(BitmFormat.DXT4_5)
    FourCC = "DXT4"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case else
    FourCC = chr(0) + chr(0) + chr(0) + chr(0)
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_RGB)
end select
select case image_info(index).format
case int32(BitmFormat.R5G6B5)
    ddpixel_RGBBitCount = 16
    RBitMask = &hF800
    GBitMask = &h7E0
    BBitMask = &h1F
    RGBAlphaBitMask = &h0
case int32(BitmFormat.A1R5G5B5)
    ddpixel_RGBBitCount = 16
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALHAPIXELS)
    RBitMask = &h7C00
    GBitMask = &h3E0
    BBitMask = &h1F

```

```

    RGBAAlphaBitMask = &h8000
case int32(BitmFormat.A4R4G4B4)
    ddpixel_RGBBitCount = 16
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &HF000
    GBitMask = &HF0
    BBitMask = &HF
    RGBAAlphaBitMask = &HF00000
case int32(BitmFormat.X8R8G8B8)
    ddpixel_RGBBitCount = 32
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &HFF000
    GBitMask = &HFF00
    BBitMask = &HFF
    RGBAAlphaBitMask = &HFF000000
case int32(BitmFormat.A8R8G8B8)
    ddpixel_RGBBitCount = 32
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &HFF0000
    GBitMask = &HFF00
    BBitMask = &HFF
    RGBAAlphaBitMask = &HFF000000
case else
    ddpixel_RGBBitCount = 0
    RBitMask = 0
    GBitMask = 0
    BBitMask = 0
    RGBAAlphaBitMask = &H0
end select

//ddsCaps
dim caps1 as integer = 0
dim caps2 as integer = 0
dim reserved_DDSCAPS() as integer
caps1 = caps1 + int32(DDSEnum.DDSCAPS_TEXTURE)
if image_info(index).num_mipmaps > 0 then
    caps1 = caps1 + int32(DDSEnum.DDSCAPS_MIPMAP)
end
if image_info(index).num_mipmaps > 0 or image_info(index).type = 2 or image_info(index).type = 3 then
    caps1 = caps1 + int32(DDSEnum.DDSCAPS_COMPLEX)
end
if image_info(index).type = 2 then
    caps2 = caps2 + int32(DDSEnum.DDSCAPS2_CUBEMAP) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEX) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEX) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEY) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEY) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEZ) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEZ)
end
if image_info(index).type = 1 then
    caps2 = caps2 + int32(DDSEnum.DDSCAPS2_VOLUME)
end
redim reserved_DDSCAPS(-1)

```

```

reserved_DDSCAPS.Append(0)
reserved_DDSCAPS.Append(0)

//write the header data
dim bw as new BinaryStream(dds)
bw.LittleEndian = true
bw.Write(dds_header)
bw.WriteInt32(size_of_structure)
bw.WriteInt32(flags)
bw.WriteInt32(height)
bw.WriteInt32(width)
bw.WriteInt32(PitchOrLinearSize)
bw.WriteInt32(depth)
bw.WriteInt32(mipmapcount)
for i as integer = 0 to UBound(reserved1)
    bw.WriteInt32(reserved1(i))
next
//the pixel format and caps are structs contained in the surface so write them then reserved2
//pixel format
bw.WriteInt32(size)
bw.WriteInt32(ddpixel_flags)
bw.Write(FourCC)
bw.WriteInt32(ddpixel_RGBBitCount)
bw.WriteInt32(RBitMask)
bw.WriteInt32(GBitMask)
bw.WriteInt32(BBitMask)
bw.WriteInt32(AlphaBitMask)
//dds caps
bw.WriteInt32(caps1)
bw.WriteInt32(caps2)
for i as integer = 0 to UBound(reserved_DDSCAPS)
    bw.WriteInt32(reserved_DDSCAPS(i))
next
bw.WriteInt32(reserved2)

//now right the raw image data
dim br as BinaryStream
if BitAnd(image_info(index).flags, &h100) = &h100 then
    br = bitmaps_f.OpenAsBinaryFile
else
    br = f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = image_info(index).offset
for i as integer = 1 to image_info(index).size step 4
    bw.WriteInt32(br.ReadInt32)
next

return dds
End Function

```

### **bitmap\_class\_RW.gradient\_colors:**

Private Function gradient\_colors(value1 as color, value2 as color) As color  
 dim temp\_color as color

```

dim r as integer = ((value1.red*2) + value2.red)/3
dim g as integer = ((value1.green*2) + value2.green)/3
dim b as integer = ((value1.blue*2) + value2.blue)/3
temp_color = rgb(r,g,b)
Return temp_color
End Function

```

### **bitmap\_class\_RW.gradient\_colors\_half:**

```

Private Function gradient_colors_half(value1 as color, value2 as color) As color
dim temp_color as color
dim R as integer = (value1.Red/2) + (value2.Red/2)
dim G as integer = (value1.Green/2) + (value2.Green/2)
dim B as integer = (value1.Blue/2) + (value2.Blue/2)
temp_color = RGB(R,G,B)
Return temp_color
End Function

```

### **bitmap\_class\_RW.inject\_DDS:**

```

Function inject_DDS(index as integer, in_data as memoryBlock, EOF as boolean = false, name as string = "Unknown")
As integer

```

```

//use the integer return to specify whether it was sucessful or failed and why

```

```

//0: Succeeded
//-1: Failed, no known reason
//-2: Failed, file size too large
//-3: Failed, bad header
//-4: Failed, image dimensions too large
//-5: Failed, image wrong format
//-6: Failed, needs to be a cubemap
//-7: Failed, needs to be a volume texture

```

```

//check to see if it's too large
if in_data.Size - &h80 > image_info(index).size then
dim temp1 as integer = in_data.Size
dim temp2 as integer = image_info(index).size
return -2
end

```

```

dim br as new BinaryStream(in_data)
br.LittleEndian = true

```

```

//check the header
br.Position = 0
dim dds_header as string
dds_header = br.read(4)
if dds_header <> "DDS " then
Return -3
end

```

```

//check the image size
br.Position = 12
dim width as int32 = br.ReadInt32
dim height as int32 = br.ReadInt32

```

```

if width > image_info(index).width or height > image_info(index).height then
    Return -4
end

//check the format
br.Position = 80
dim ddpixel_flags as int32 = br.ReadInt32
dim FourCC as string = br.Read(4)
if BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_FOURCC)) = int32(DDSEnum.DDPF_FOURCC) then
    //dxt format
    select case FourCC
    case "DXT1"
        if image_info(index).format <> int32(BitmFormat.DXT1) then Return - 5

    case "DXT2", "DXT3"
        if image_info(index).format <> int32(BitmFormat.DXT2_3) then Return - 5

    case "DXT4", "DXT5"
        if image_info(index).format <> int32(BitmFormat.DXT4_5) then Return - 5

    end select
else
    //check to make sure the RGB format is correct
    br.Position = 88
    dim RGBbitcount as integer = br.ReadInt32
    dim Rbitmask as integer = br.ReadInt32
    dim Gbitmask as integer = br.ReadInt32
    dim Bbitmask as integer = br.ReadInt32
    dim RGBAlphaBitMask as integer = br.ReadInt32

    select case image_info(index).format
    case int32(BitmFormat.R5G6B5)
        if RGBbitcount <> 16 or _
            RBitMask <> &hF800 or _
            GBitMask <> &h7E0 or _
            BBitMask <> &h1F or _
            RGBAlphaBitMask <> &h0 then return -5
    case int32(BitmFormat.A1R5G5B5)
        if RGBbitcount <> 16 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _
            RBitMask <> &h7C00 or _
            GBitMask <> &h3E0 or _
            BBitMask <> &h1F or _
            RGBAlphaBitMask <> &h8000 then return -5
    case int32(BitmFormat.A4R4G4B4)
        if RGBbitcount <> 16 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _
            RBitMask <> &HF000 or _
            GBitMask <> &HF0 or _
            BBitMask <> &HF or _
            RGBAlphaBitMask <> &HF00000 then return -5
    case int32(BitmFormat.X8R8G8B8)
        //won't hold alpha channel against you
        if RGBbitcount <> 32 or _

```

```

        RBitMask <> &HFF000 or _
        GBitMask <> &HFF00 or _
        BBitMask <> &HFF then return -5
    case int32(BitmFormat.A8R8G8B8)
        if RGBbitcount <> 32 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _
            RBitMask <> &HFF0000 or _
            GBitMask <> &HFF00 or _
            BBitMask <> &HFF or _
            RGBAlphaBitMask <> &HFF000000 then return -5
        end select
    end
end

//check for cubemappery
if image_info(index).type = 2 then
    br.position = 112
    dim ddscaps2 as int32 = br.readint32
    if BitAnd(ddscaps2, int32(DDSEnum.DDSCAPS2_CUBEMAP)) <> int32(DDSEnum.DDSCAPS2_CUBEMAP) then
        Return -6
    end
end

//volume checking
if image_info(index).type = 1 then
    br.position = 112
    dim ddscaps2 as int32 = br.readint32
    if BitAnd(ddscaps2, int32(DDSEnum.DDSCAPS2_VOLUME)) <> int32(DDSEnum.DDSCAPS2_VOLUME) then
        Return -7
    end
    br.position = 8
    dim dds_flags as int32 = br.readint32
    if BitAnd(dds_flags, int32(DDSEnum.DDSD_DEPTH)) <> int32(DDSEnum.DDSD_HEIGHT) then
        Return -7
    end
end

end

//write the data
//maybe later add an option to inject into raw data?
if not EOF then
    dim bw as BinaryStream
    if BitAnd(image_info(index).flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(true)
    else
        bw = f.OpenAsBinaryFile(true)
    end
    bw.LittleEndian = true
    bw.Position = image_info(index).offset
    br.Position = &h80
    for i as integer = 1 to in_data.Size - &h80 step 4
        bw.WriteInt32(br.ReadInt32)
    next
    bw.close
    br.close

    //update the info in the map

```

```

    bw = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = image_info(index).position + 4
    bw.Writeshort(width)
    bw.Writeshort(height)
    bw.Position = image_info(index).position + 28
    bw.WriteInt32(in_data.Size - &h80)
    bw.Close
end
if EOF AND (BitAnd(image_info(index).flags, &h100) = &h100) then
    dim bitmap_map as new H1SupportMap.SupportMap
    bitmap_map.read(bitmaps_f)

    //ewww all this to truncate 80 bytes
    dim inject_data as new MemoryBlock(0)
    dim bw_data as new BinaryStream(inject_data)
    bw_data.LittleEndian = true
    dim br_data as new BinaryStream(in_data)
    br_data.LittleEndian = true
    br.Position = &h80
    for i as integer = 1 to in_data.Size - &h80 step 4
        bw_data.WriteInt32(br_data.ReadInt32)
    next
    bw_data.close
    br_data.close

    image_info(index).offset = bitmap_map.inject(inject_data, name)
    //make sure it's now part of bitmaps.map
    if BitAnd(image_info(index).flags, &h100) <> &h100 then
        image_info(index).flags = image_info(index).flags + &h100
    end
    //update the info in the map
    dim bw as BinaryStream
    bw = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = image_info(index).position + 4
    bw.Writeshort(width)
    bw.Writeshort(height)
    bw.Position = image_info(index).position + 28
    bw.WriteInt32(in_data.Size - &h80)
    bw.Position = image_info(index).position + 14 //flags offset
    bw.WriteInt32(image_info(index).flags)
    bw.Position = image_info(index).position + 24 //write the updated offset
    bw.WriteInt32(offset)
    bw.Close
end

```

```

    return 0
End Function

```

### **bitmap\_class\_RW.read:**

```

Sub read()
    dim br as BinaryStream = f.OpenAsBinaryFile

```

```

br.LittleEndian = true
br.Position = offset
header.read(br, magic)

br.Position = header.image_offset
redim images(-1)
redim image_info(-1)
for i as integer = 1 to header.image_count
    dim temp_image as new bitm_image_info
    temp_image.read(br)
    image_info.Append( temp_image )
next
End Sub

```

### **bitmap\_class\_RW.read\_2D:**

```

Private Sub read_2D(info as bitm_image_info)
    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_pic as Picture
        temp_pic = read_A8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_A8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_image.mip_maps.append(temp_mip_image)
        next
        images.Append temp_image

    case &h1 //Y8
        dim temp_pic as Picture
        temp_pic = read_Y8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_Y8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_image.mip_maps.append(temp_mip_image)
        next
        images.Append temp_image

    case &h2 //AY8
        dim temp_pixs() as Picture
        temp_pixs = read_AY8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))

```



```

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_AY8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h3 //A8Y8
    dim temp_pixs() as Picture
    temp_pixs = read_A8Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h6 //R5G6B5
    dim temp_pic as Picture
    temp_pic = read_R5G6B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_R5G6B5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h8 //A1R5G5B5
    dim temp_pixs() as Picture
    temp_pixs = read_A1R5G5B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h9 //A4R4G4B4
    dim temp_pixs() as Picture
    temp_pixs = read_A4R4G4B4(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next

```

```

images.Append temp_image

case &hA //X8R8G8B8
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hB //A8R8G8B8
    dim temp_pixs() as Picture
    temp_pixs = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hE //DXT1
    dim temp_pic as Picture
    temp_pic = read_DXT1(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_DXT1(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hF //DXT2and3
    dim temp_pixs() as Picture
    temp_pixs = read_DXT2_3(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h10 //DXT4and5
    dim temp_pixs() as Picture
    temp_pixs = read_DXT4_5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))

```

```

    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h11 //P8
    dim temp_pic as Picture
    temp_pic = read_P8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_P8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

end select
End Sub

bitmap_class_RW.read_3D:
Private Sub read_3D(info as bitm_image_info)
    //so it's basically of the a similar format but unlike cubemaps, it reads each layer then mips
    //also, mips have half as many layers as the main, halving until there's only 1 layer per mip level

    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_pic as Picture
        temp_pic = read_A8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        dim layers(-1) as picture
        for i as integer = 2 to info.depth
            layers.append read_A8(info.width, info.height, br)
        next
        temp_image.add_layers(layers)

        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_A8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)

            dim mip_layers(-1) as Picture

```

```

    for j as integer = 2 to info.depth/(2^k)
        mip_layers.append read_A8(info.width/(2^k), info.height/(2^k), br)
    next
    temp_mip_image.add_layers(mip_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h1 //Y8
    dim temp_pic as Picture
    temp_pic = read_Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_Y8(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_Y8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)

        dim mip_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            mip_layers.append read_Y8(info.width/(2^k), info.height/(2^k), br)
        next
        temp_mip_image.add_layers(mip_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h2 //AY8
    dim temp_pixs() as Picture
    temp_pixs = read_AY8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_AY8(info.width, info.height, br)
        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_AY8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

        dim mip_layers(-1), mip_alpha_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)

```

```

        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_AY8(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h3 //A8Y8
    dim temp_pixs() as Picture
    temp_pixs = read_A8Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A8Y8(info.width, info.height, br)
        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

        dim mip_layers(-1), mip_alpha_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            dim temp_layer_pixs(-1) as picture
            temp_layer_pixs = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
            mip_layers.append temp_layer_pixs(1)
            mip_alpha_layers.append temp_layer_pixs(0)
        next
        temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h6 //R5G6B5
    dim temp_pic as Picture
    temp_pic = read_R5G6B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_R5G6B5(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture

```

```

temp_mip = read_R5G6B5(info.width/(2^k), info.height/(2^k), br)
dim temp_mip_image as new bitm_image(temp_mip)

dim mip_layers(-1) as Picture
for j as integer = 2 to info.depth/(2^k)
    mip_layers.append read_R5G6B5(info.width/(2^k), info.height/(2^k), br)
next
temp_mip_image.add_layers(mip_layers)

temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h8 //A1R5G5B5
dim temp_pixs() as Picture
temp_pixs = read_A1R5G5B5(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A1R5G5B5(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h9 //A4R4G4B4
dim temp_pixs() as Picture
temp_pixs = read_A4R4G4B4(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A4R4G4B4(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)

```

```

next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hA //X8R8G8B8
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_X8R8G8B8(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)

        dim mip_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            mip_layers.append read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        next
        temp_mip_image.add_layers(mip_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hB //A8R8G8B8
    dim temp_pixs() as Picture
    temp_pixs = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A8R8G8B8(info.width, info.height, br)

```

```

        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hE //DXT1
    dim temp_pic as Picture
    temp_pic = read_DXT1(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_DXT1(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_DXT1(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)

        dim mip_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            mip_layers.append read_DXT1(info.width/(2^k), info.height/(2^k), br)
        next
        temp_mip_image.add_layers(mip_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hF //DXT2and3
    dim temp_pixs() as Picture
    temp_pixs = read_DXT2_3(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth

```



```

    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_DXT2_3(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h10 //DXT4and5
    dim temp_pixs() as Picture
    temp_pixs = read_DXT4_5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_DXT4_5(info.width, info.height, br)
        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

        dim mip_layers(-1), mip_alpha_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            dim temp_layer_pixs(-1) as picture
            temp_layer_pixs = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
            mip_layers.append temp_layer_pixs(1)
            mip_alpha_layers.append temp_layer_pixs(0)
        next
        temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next

```

```

images.Append temp_image

case &h11 //P8
dim temp_pic as Picture
temp_pic = read_P8(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pic)
dim layers(-1) as picture
for i as integer = 2 to info.depth
    layers.append read_P8(info.width, info.height, br)
next
temp_image.add_layers(layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mip as Picture
    temp_mip = read_P8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mip)

    dim mip_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        mip_layers.append read_P8(info.width/(2^k), info.height/(2^k), br)
    next
    temp_mip_image.add_layers(mip_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

end select
End Sub

bitmap_class_RW.read_A1R5G5B5:
Private Function read_A1R5G5B5(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(Width, Height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint16 = br.readuint16
            dim A as UInt16 = Bitwise.ShiftRight(value, 15) * &hFF
            dim R as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 10), &h1F) * &hFF)/31
            dim G as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 5), &h1F) * &hFF)/31
            dim B as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 0), &h1F) * &hFF)/31
            p_color.RGBSurface.pixel(x,y) = RGB(r,g,b)
            p_alpha.RGBSurface.pixel(x,y) = RGB(a,a,a)
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

**bitmap\_class\_RW.read\_A4R4G4B4:**

Private Function read\_A4R4G4B4(width as integer, height as integer, br as binaryStream) As picture()

```
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint16 = br.readuint16
            dim A as integer = (Bitwise.ShiftRight(value, 12) * &hFF) / 15
            dim R as integer = (BitAnd(Bitwise.ShiftRight(value, 8), &h0F)*&hFF)/15
            dim G as integer = (BitAnd(Bitwise.ShiftRight(value, 4), &h0F)*&hFF)/15
            dim B as integer = (BitAnd(Bitwise.ShiftRight(value, 0), &h0F)*&hFF)/15
            dim temp_color as color = RGB(R, G, B)
            p_color.RGBSurface.pixel(X,Y) = temp_color
            p_alpha.RGBSurface.pixel(X,Y) = RGB(A,A,A)
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function
```

**bitmap\_class\_RW.read\_A8:**

Private Function read\_A8(width as integer, height as integer, br as binaryStream) As picture

```
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            //value = Bitwise.ShiftLeft(value, 24)
            dim temp_color as color = RGB(value, value, value)
            p.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p
End Function
```

**bitmap\_class\_RW.read\_A8R8G8B8:**

Private Function read\_A8R8G8B8(width as integer, height as integer, br as binaryStream) As picture()

```
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint32 = br.readuint32
            dim A as integer = Bitwise.ShiftRight(value, 24, 32)
            dim R as integer = BitAnd(Bitwise.ShiftRight(value, 16, 32), 255)
```

```

        dim G as integer = BitAnd(Bitwise.ShiftRight(value, 8, 32), 255)
        dim B as integer = BitAnd(Bitwise.ShiftRight(value, 0, 32), 255)
        //this might just be a fancy way of reading each channel
        //as uint8
        dim temp_color as color = RGB(R, G, B)
        p_color.RGBSurface.pixel(X,Y) = temp_color
        p_alpha.RGBSurface.pixel(X,Y) = RGB(A,A,A)
    next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

bitmap_class_RW.read_A8Y8:
Private Function read_A8Y8(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim r_g_b as uint8 = Bitwise.ShiftRight(value, 8)
            dim alpha as uint8 = BitAnd(value, &hFF)
            p_color.RGBSurface.pixel(X,Y) = RGB(r_g_b, r_g_b, r_g_b)
            p_alpha.RGBSurface.pixel(X,Y) = RGB(alpha, alpha, alpha)
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

```

bitmap_class_RW.read_AY8:
Private Function read_AY8(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim temp_color as color = RGB(value, value, value)
            p_color.RGBSurface.pixel(X,Y) = temp_color
            p_alpha.RGBSurface.pixel(X,Y) = temp_color
        next
    next
next

```

```

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

### **bitmap\_class\_RW.read\_bitmap:**

```

Sub read_bitmap(index as integer)
    redim images(-1)
    redim cube_images(-1)

    select case image_info(index).type
    case 0
        read_2D(image_info(index))
    case 1
        read_3D(image_info(index))
    case 2
        read_cube_maps(image_info(index))
    end select
End Sub

```

### **bitmap\_class\_RW.read\_bitmaps:**

```

Sub read_bitmaps()
    redim images(-1)
    redim cube_images(-1)

    for i as integer = 0 to UBound(image_info)
        select case image_info(i).type
        case 0
            read_2D(image_info(i))
        case 1
            read_3D(image_info(i))
        case 2
            read_cube_maps(image_info(i))
        end select
    next
End Sub

```

### **bitmap\_class\_RW.read\_cube\_maps:**

```

Private Sub read_cube_maps(info as bitm_image_info)
    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_images() as bitm_image

```

```

for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pic as Picture
    temp_pic = read_A8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_A8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h1 //Y8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_Y8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_Y8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h2 //AY8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pix() as Picture
        temp_pix = read_AY8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_AY8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next

```

```

next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h3 //A8Y8
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_A8Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A8Y8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h6 //R5G6B5
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pic as Picture
    temp_pic = read_R5G6B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_R5G6B5(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h8 //A1R5G5B5
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_A1R5G5B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps

```

```

    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A1R5G5B5(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h9 //A4R4G4B4
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_A4R4G4B4(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A4R4G4B4(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hA //X8R8G8B8
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_X8R8G8B8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hB //A8R8G8B8
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture

```



```

    temp_pix = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A8R8G8B8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hE //DXT1
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_DXT1(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_DXT1(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &hF //DXT2and3
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pix() as Picture
        temp_pix = read_DXT2_3(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_DXT2_3(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)

```

```

cube_images.Append temp_cube

case &h10 //DXT4and5
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pix() as Picture
        temp_pix = read_DXT4_5(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_DXT4_5(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h11 //P8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_P8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_P8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

end select
End Sub

```

### **bitmap\_class\_RW.read\_DXT1:**

```

Private Function read_DXT1(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = newPicture(width, height, 32)

    dim color1,color2,color3,color4 as color
    dim zero_color as Color = RGB(0,0,0)
    dim chunkspersHline as integer = Width/4

```

```
if chunksperHline = 0 then chunksperHline = 1
```

```
dim i as integer = 0
```

```
while i < width*height
```

```
    dim c1, c2 as UInt16
```

```
    dim trans as boolean
```

```
    c1 = br.ReadUInt16
```

```
    c2 = br.ReadUInt16
```

```
    if c1 > c2 then
```

```
        trans = false
```

```
    else
```

```
        trans = true
```

```
    end
```

```
    color1 = uint16_to_color(c1)
```

```
    color2 = uint16_to_color(c2)
```

```
    if not trans then
```

```
        color3 = gradient_colors(color1, color2)
```

```
        color4 = gradient_colors(color2, color1)
```

```
    else
```

```
        color3 = gradient_colors_half(color1, color2)
```

```
        color4 = zero_color
```

```
    end
```

```
dim cdata as UInt64 = br.ReadUInt32
```

```
dim ChunkNum as UInt32 = i / 16
```

```
dim XPos as UInt32 = ChunkNum mod chunksperHline
```

```
dim YPos as UInt32 = (ChunkNum - XPos) / chunksperHline
```

```
dim sizeW, sizeH as integer
```

```
if Height < 4 then
```

```
    sizeH = Height
```

```
else
```

```
    sizeH = 4
```

```
end
```

```
if Width < 4 then
```

```
    sizeW = Width
```

```
else
```

```
    sizeW = 4
```

```
end
```

```
for y as integer = 0 to sizeH-1
```

```
    for x as integer = 0 to sizeW-1
```

```
        dim temp_color as color
```

```
        select case BitAnd(cdata, &h3)
```

```
            case 0
```

```
                temp_color = color1
```

```
            case 1
```

```
                temp_color = color2
```

```
            case 2
```

```
                temp_color = color3
```

```
            case 3
```

```
                temp_color = color4
```

```

        case else
            break
        end select
        p.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = temp_color
        cdata = Bitwise.ShiftRight(cdata, 2)
    next
next

    i = i + 16
wend

Return p
End Function

```

### **bitmap\_class\_RW.read\_DXT2\_3:**

Private Function read\_DXT2\_3(width as integer, height as integer, br as binaryStream) As picture()

```

    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    dim chunkspersHline as integer = width/4
    if chunkspersHline = 0 then chunkspersHline = 1

    for i as integer = 0 to (width*height)-1 step 16
        dim Alpha(-1) as UInt16
        for j as integer = 0 to 3
            Alpha.append br.ReadUInt16
        next
        dim colors(-1) as color
        dim c1 as uint16 = br.readuint16
        dim c2 as uint16 = br.readuint16
        if c2 >= c1 then
            //dim temp as integer <- for breakpoints
        end
        colors.Append uint16_to_color(c1)
        colors.Append uint16_to_color(c2)
        colors.Append gradient_colors(colors(0), colors(1))
        colors.Append gradient_colors(colors(1), colors(0))
        dim value as UInt32 = br.ReadUInt32
        dim ChunkNum as UInt32 = i/16
        dim XPos as UInt32 = ChunkNum mod chunkspersHline
        dim YPos as UInt32 = (ChunkNum - XPos)/chunkspersHline
        dim sizeh, sizew as integer
        if height < 4 then sizeh = height else sizeh = 4
        if width < 4 then sizew = width else sizew = 4

        for y as integer = 0 to sizeh - 1
            for x as integer = 0 to sizew - 1
                dim temp_color as color = colors(BitAnd(value, &h03))
                value = Bitwise.ShiftRight(value, 2)
                dim alpha_value as integer = BitAnd(Alpha(y), &h0F) * 17
                dim alpha_color as color = RGB(alpha_value, alpha_value, alpha_value)
                Alpha(y) = Bitwise.ShiftRight(Alpha(y), 4)
            next
        next
    next

```

```

        p_color.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = temp_color
        p_alpha.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = alpha_color
    next
next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_RW.read\_DXT4\_5:**

```

Private Function read_DXT4_5(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    dim chunkspersHline as integer = width/4
    if chunkspersHline = 0 then chunkspersHline = 1

    for i as integer = 0 to (width*height) - 1 step 16
        dim alpha(7) as UInt8
        alpha(0) = br.ReadUInt8
        alpha(1) = br.ReadUInt8
        dim temp_word as UInt16 = br.ReadUInt16
        dim temp_dword as UInt32 = br.ReadUInt32
        dim AlphaDat as UInt32 = BitOr(temp_word, Bitwise.ShiftRight(temp_dword, 16)) //might need to alter this due
        to endianness
        if alpha(0) > alpha(1) then
            //8-alpha block: derive the other size alphas
            //Bit code 000 = alpha_0, 001 = alpha_1, others are interpolated.
            Alpha(2) = (6 * Alpha(0) + 1 * Alpha(1) + 3) / 7 // bit code 010
            Alpha(3) = (5 * Alpha(0) + 2 * Alpha(1) + 3) / 7 // bit code 011
            Alpha(4) = (4 * Alpha(0) + 3 * Alpha(1) + 3) / 7 // bit code 100
            Alpha(5) = (3 * Alpha(0) + 4 * Alpha(1) + 3) / 7 // bit code 101
            Alpha(6) = (2 * Alpha(0) + 5 * Alpha(1) + 3) / 7 // bit code 110
            Alpha(7) = (1 * Alpha(0) + 6 * Alpha(1) + 3) / 7 // bit code 111
        else
            //6-alpha block.
            //Bit code 000 = alpha_0, 001 = alpha_1, others are interpolated.
            Alpha(2) = (4 * Alpha(0) + 1 * Alpha(1) + 2) / 5 // Bit code 010
            Alpha(3) = (3 * Alpha(0) + 2 * Alpha(1) + 2) / 5 // Bit code 011
            Alpha(4) = (2 * Alpha(0) + 3 * Alpha(1) + 2) / 5 // Bit code 100
            Alpha(5) = (1 * Alpha(0) + 4 * Alpha(1) + 2) / 5 // Bit code 101
            Alpha(6) = 0 // Bit code 110
            Alpha(7) = 255 // Bit code 111
        end
        dim colors(-1) as color
        colors.Append(uint16_to_color(br.ReadUInt16))
        colors.Append(uint16_to_color(br.ReadUInt16))
        colors.Append(gradient_colors(colors(0), colors(1)))
        colors.Append(gradient_colors(colors(1), colors(0)))
    next
end

```

```

dim value as UInt32 = br.ReadUInt32

dim ChunkNum as UInt32 = i/16
dim XPos as UInt32 = ChunkNum mod chunkspersHline
dim YPos as UInt32 = (ChunkNum - XPos)/chunkspersHline
dim sizeh, sizew as integer
if height < 4 then sizeh = height else sizeh = 4
if width < 4 then sizew = width else sizew = 4

for x as integer = 0 to sizeh -1
    for y as integer = 0 to sizew - 1
        dim temp_color as color = colors(BitAnd(value, &h03))
        value = Bitwise.ShiftRight(value, 2)
        dim alpha_value as UInt8 = alpha(BitAnd(AlphaDat, &h07))
        AlphaDat = Bitwise.ShiftRight(AlphaDat, 3)
        p_color.RGBSurface.pixel(y+(XPos*4), x+(YPos*4)) = temp_color
        p_alpha.RGBSurface.pixel(y+(XPos*4), x+(YPos*4)) = RGB(alpha_value, alpha_value, alpha_value)
    next
next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_RW.read\_P8:**

```

Private Function read_P8(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim temp_color as color = RGB(value, value, value)
            p.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p
End Function

```

### **bitmap\_class\_RW.read\_R5G6B5:**

```

Private Function read_R5G6B5(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint16 = br.readuint16
            dim temp_color as color = uint16_to_color(value)
            p_color.RGBSurface.pixel(X,Y) = temp_color
        next
    next

```

```

    next
next

    return p_color
End Function

```

### **bitmap\_class\_RW.read\_X8R8G8B8:**

```

Private Function read_X8R8G8B8(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint32 = br.readuint32
            dim R as integer = BitAnd(Bitwise.ShiftRight(value, 16, 32), 255)
            dim G as integer = BitAnd(Bitwise.ShiftRight(value, 8, 32), 255)
            dim B as integer = BitAnd(Bitwise.ShiftRight(value, 0, 32), 255)
            //this might just be a fancy way of reading each channel
            //as uint8
            dim temp_color as color = RGB(R, G, B)
            p_color.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p_color
End Function

```

### **bitmap\_class\_RW.read\_Y8:**

```

Private Function read_Y8(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim temp_color as color = RGB(value, value, value)
            p.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p
End Function

```

### **bitmap\_class\_RW.uint16\_to\_color:**

```

Private Function uint16_to_color(input as uint16) As color
    dim temp_color as color
    dim R as integer = (BitAnd(Bitwise.ShiftRight(input, 11), &h1F) * &hFF)/31
    dim G as integer = (BitAnd(Bitwise.ShiftRight(input, 5), &h3F) * &hFF)/63
    dim B as integer = (BitAnd(Bitwise.ShiftRight(input, 0), &h1F) * &hFF)/31
    temp_color = RGB(R,G,B)
    return temp_color
End Function

```

bitmap\_class\_RW.update\_offset:

Function update\_offset(image\_info\_ind as integer, new\_offset as integer, internal as boolean) As boolean

if image\_info\_ind < 0 OR image\_info\_ind > UBound(image\_info) then Return False

image\_info(image\_info\_ind).offset = new\_offset

if BitAnd(image\_info(image\_info\_ind).flags, &h100) = &h100 then

//external

if internal then

image\_info(image\_info\_ind).flags = BitXor(image\_info(image\_info\_ind).flags, &h100)

end

else

//internal

if not internal then

image\_info(image\_info\_ind).flags = BitXor(image\_info(image\_info\_ind).flags, &h100)

end

end

dim bw as BinaryStream = f.OpenAsBinaryFile(true)

bw.LittleEndian = true

image\_info(image\_info\_ind).write(bw)

Return True

End Function

**bitmap\_class\_RW.update\_offset:**

Function update\_offset(byref bw as binarystream, image\_info\_ind as integer, new\_offset as integer, internal as boolean) As boolean

if image\_info\_ind < 0 OR image\_info\_ind > UBound(image\_info) then Return False

image\_info(image\_info\_ind).offset = new\_offset

if BitAnd(image\_info(image\_info\_ind).flags, &h100) = &h100 then

//external

if internal then

image\_info(image\_info\_ind).flags = BitXor(image\_info(image\_info\_ind).flags, &h100)

end

else

//internal

if not internal then

image\_info(image\_info\_ind).flags = BitXor(image\_info(image\_info\_ind).flags, &h100)

end

end

bw.LittleEndian = true

image\_info(image\_info\_ind).write(bw)

Return True

End Function

bitmaps\_f As folderItem

cube\_images(-1) As cube\_map

f As folderItem

header As bitm\_header\_RW



images(-1) As bitm\_image

image\_info(-1) As bitm\_image\_info

magic As Integer

offset As Integer

DDS resource

[http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9\\_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp](http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp)

End Class

## **Class bitm\_header RW**

### **bitm\_header\_RW.read:**

Sub read(byref br as binaryStream, magic as integer)

    br.LittleEndian = true

    redim unknown(-1)

    for i as integer = 1 to 22

        unknown.append br.ReadInt32

    next

    offset\_to\_first = br.ReadInt32 - magic

    unknown23 = br.ReadInt32

    image\_count = br.ReadInt32

    image\_offset = br.ReadInt32 - magic

    unknown25 = br.ReadInt32

End Sub

image\_count As Integer

image\_offset As Integer

offset\_to\_first As Integer

unknown() As Integer

unknown23 As Integer

unknown25 As Integer

End Class

## **Class snd\_class EX**

### **snd\_class\_EX.Constructor:**

Sub Constructor(in\_data as memoryblock, in\_offset as integer, in\_magic as integer)

```

    data_ref = in_data
    offset = in_offset
    magic = in_magic
End Sub

```

### **snd\_class\_EX.read:**

```

Sub read()
    dim br as BinaryStream = new BinaryStream(data_ref)
    br.LittleEndian = true
    br.Position = 0
    header = new snd_header_EX
    header.read(br, magic, offset)

    br.position = header.pitch_translation
    redim pitch(-1)
    for i as integer = 1 to header.pitch_count
        dim temp_pitch as new snd_pitch_EX
        temp_pitch.read(br, magic, offset)
        pitch.Append temp_pitch
    next
End Sub

```

### **snd\_class\_EX.update\_offset:**

Function update\_offset(pitch\_ind as integer, perm\_ind as integer, new\_offset as integer, internal as boolean) As boolean

```

    if pitch_ind < 0 OR pitch_ind > UBound(pitch) then Return False
    if perm_ind < 0 or perm_ind > UBound(pitch(pitch_ind).permutations) then Return False

    pitch(pitch_ind).permutations(perm_ind).offset = new_offset
    pitch(pitch_ind).permutations(perm_ind).internal = internal
    dim bw as BinaryStream = new BinaryStream(data_ref)
    bw.LittleEndian = true
    pitch(pitch_ind).permutations(perm_ind).write(bw)

    Return True
End Function

```

### **snd\_class\_EX.update\_offset:**

Function update\_offset(byref bw as binarystream, pitch\_ind as integer, perm\_ind as integer, new\_offset as integer, internal as boolean) As boolean

```

    if pitch_ind < 0 OR pitch_ind > UBound(pitch) then Return False
    if perm_ind < 0 or perm_ind > UBound(pitch(pitch_ind).permutations) then Return False

    pitch(pitch_ind).permutations(perm_ind).offset = new_offset
    pitch(pitch_ind).permutations(perm_ind).internal = internal
    bw.LittleEndian = true
    pitch(pitch_ind).permutations(perm_ind).write(bw)

    Return True
End Function
data_ref As memoryBlock

```

```

header As snd_header_EX

```

magic As Integer

offset As Integer

pitch(-1) As snd\_pitch\_EX

End Class

## **Class snd\_header\_EX**

### **snd\_header\_EX.read:**

Sub read(byref br as binaryStream, magic as integer, offset as integer)

```
    br.littleendian = true
    dim pos as integer = br.Position
    dim temp_int as integer = br.ReadUInt32
    if Bitwise.BitAnd(temp_int, 1) = 1 then
        adpcm_blocksize = true
    else
        adpcm_blocksize = false
    end
    if Bitwise.BitAnd(temp_int, 2) = 2 then
        split_into_permutations = true
    else
        split_into_permutations = false
    end
```

```
    br.Position = pos + &h6
    temp_int = br.ReadUInt16
    select case temp_int
    case 0
        sample_rate = 22
    case 1
        sample_rate = 44
    case else
        sample_rate = 0
    end select
```

```
    br.Position = pos + &h6c
    temp_int = br.ReadUInt16
    select case temp_int
    case 1
        stereo = true
    case else
        stereo = false
    end select
```

```
    br.Position = pos + &h6e
    temp_int = br.ReadUInt16
    compression = temp_int
```

```
    br.Position = pos + &h98
    pitch_count = br.ReadUInt32
```

```
pitch_translation = br.ReadUInt32 - magic - offset
```

```
br.Position = pos + 164  
End Sub  
adpcm_blocksize As boolean
```

### **snd\_header\_EX.compression:**

```
compression As Integer
```

```
//0 = none
```

```
//1 = xbox adpcm
```

```
//2 = IMA adpcm
```

```
//3 = ogg vorbis
```

```
pitch_count As Integer
```

```
pitch_translation As Integer
```

```
sample_rate As Integer
```

```
split_into_permutations As boolean
```

```
stereo As boolean
```

```
End Class
```

## **Class snd\_pitch\_EX**

### **snd\_pitch\_EX.read:**

```
Sub read(byref br as binaryStream, magic as integer, offset as integer)
```

```
br.LittleEndian = true
```

```
dim pos as integer = br.Position
```

```
br.Position = pos + &h3c
```

```
perm_count = br.ReadUInt32
```

```
perm_translation = br.ReadUInt32 - magic - offset
```

```
br.Position = pos + &h2c
```

```
track_count = br.ReadUInt16
```

```
br.position = perm_translation
```

```
redim permutations(-1)
```

```
for i as integer = 1 to perm_count
```

```
dim temp_perm as new snd_perm
```

```
temp_perm.read(br, magic)
```

```
permutations.Append temp_perm
```

```
next
```

```
br.Position = pos + 72
```

```
//problems related to track stuff
```

```
'redim tracks(-1)
```

```
'dim current_index as integer = 0
```

```
'dim previous_index as integer = 0
```

```

'dim index_ar(-1) as integer
'for i as integer = 0 to UBound(permutations)
'index_ar.append i
'next
'while previous_index <= permutations.ubound AND current_index <= permutations.Ubound AND current_index
<> -1
'dim start as Boolean = true
'dim temp_track as new snd_track
'while current_index <> -1
'temp_track.perm_list.Append current_index
'index_ar(current_index) = -1
'previous_index = current_index
'if permutations(current_index).next_perm = current_index then
'current_index = -1
'else
'current_index = permutations(current_index).next_perm
'end
'start = false
'wend
'tracks.Append temp_track
'for i as integer = 0 to UBound(index_ar)
'if index_ar(i) <> -1 then
'current_index = index_ar(i)
'exit for i
'end
'next
'wend
'

'if UBound(tracks) +1 <> track_count then
'dim temp as integer //for break point
'end
'for i as integer = 0 to UBound(index_ar)
'if index_ar(i) <> -1 then
'dim temp2 as integer
'end
'next
End Sub
permutations(-1) As snd_perm

perm_count As Integer

perm_translation As Integer

tracks(-1) As snd_track

track_count As Integer

End Class

```

## **Class snd\_perm**

**snd\_perm.read:**

```

Sub read(byref br as binaryStream, magic as integer)
    br.LittleEndian = true
    pos = br.Position
    br.Position = pos + &h28
    compression = br.ReadUInt16

    br.Position = pos + &h40
    size = br.ReadUInt32

    br.Position = pos + &h44
    flag = br.ReadInt8
    if flag = 0 then
        internal = true
    else
        internal = false
    end

    br.Position = pos + &h48
    offset = br.ReadUInt32

    br.Position = pos + &h2a
    next_perm = br.ReadInt16

    br.Position = pos + 124
End Sub

```

### **snd\_perm.write:**

```

Sub write(byref bw as binaryStream)
    bw.LittleEndian = true
    bw.Position = pos

    bw.Position = pos + &h28
    bw.WriteUInt16(compression)

    bw.Position = pos + &h40
    bw.WriteUInt32(size)

    bw.Position = pos + &h44
    if internal then
        flag = 0
    else
        flag = 1
    end
    bw.WriteInt8(flag)

    bw.Position = pos + &h48
    bw.WriteUInt32(offset)

    bw.Position = pos + &h2a
    bw.WriteInt16(next_perm)

    bw.Position = pos + 124
End Sub

```

snd\_perm.compression:

compression As Integer

//0 = none

//1 = xbox adpcm

//2 = IMA adpcm

//3 = ogg vorbis

flag As Integer

internal As boolean

next\_perm As Integer

offset As Integer

pos As Integer

size As Integer

End Class

## **Class snd\_track**

### **snd\_track.Constructor:**

Sub Constructor()

    redim perm\_list(-1)

End Sub

perm\_list() As Integer

End Class

## **Class ogg\_header**

### **ogg\_header.read:**

Sub read(byref br as binaryStream)

    redim segment\_table(-1)

    br.LittleEndian = true

    offset = br.Position

    capture\_pattern = br.Read(4)

    version = br.ReadInt8

    header\_type = br.ReadInt8

    granule\_position = br.ReadInt64

    bitstream\_serial\_number = br.ReadInt32

    page\_sequence\_number = br.ReadInt32

    checksum = br.ReadUInt32

    page\_segments = br.ReadUInt8

    for i as integer = 1 to page\_segments

        segment\_table.Append br.ReadUInt8

    next

```

    header_size = br.Position - offset
    data_size = 0
    for i as integer = 0 to UBound(segment_table)
        data_size = data_size + segment_table(i)
    next
End Sub
bitstream_serial_number As Integer

capture_pattern As string

checksum As uint32

data_size As Integer

granule_position As Integer

header_size As Integer

header_type As Integer

offset As Integer

page_segments As Integer

page_sequence_number As Integer

segment_table() As Integer

version As Integer

End Class

```

## **Class map\_control\_RW**

Inherits ContainerControl

### **map\_control\_RW.Constructor:**

```

Sub Constructor(byref in_w as main_window, in_map_index as integer)
    main = in_w
    map_index = in_map_index

    file_edit.text = main.maps_lite(map_index).fs.absolutepath

    select case main.maps_lite(map_index).header.version
    case 609 //CE
        game_text.Text = "Game: Halo Custom Edition"
    case 5 //xbox
        game_text.Text = "Game: Halo (Xbox)"
    case 6 //demo
        game_text.Text = "Game: Halo Demo/Trial"
    end select
End Sub

```



```

case 7 //full
    game_text.Text = "Game: Halo (PC/Mac)"
case else
    game_text.Text = "Game: Unknown"
end select

select case main.maps_lite(map_index).header.maptype
case 0
    maptype_text.Text = "Map Type: Campaign"
case 1
    maptype_text.Text = "Map Type: Multiplayer"
case 2
    maptype_text.Text = "Map Type: User Interface"
case else
    maptype_text.Text = "Map Type: Unknown"
end select

name_text.text = "Name: " + main.maps_lite(map_index).header.name.replaceAll(chr(0), "")
build_text.text = "Build: " + main.maps_lite(map_index).header.builddate.replaceAll(chr(0), "")
size_text.Text = "Map Size: 0x" + hex(main.maps_lite(map_index).header.decomp_len)
primary_edit.text = "0x" + hex(main.maps_lite(map_index).index_header.index_magic)
map_edit.text = "0x" + hex(main.maps_lite(map_index).magic)
metasize_text.text = "Meta Size: 0x" + hex(main.maps_lite(map_index).header.TagIndexMetaLength)
tagcount_text.text = "Tag Count: " + str(main.maps_lite(map_index).index_header.tagcount)
vertsize_text.text = "Vertex Data Size: 0x" + _
hex(main.maps_lite(map_index).index_header.ind_offset - _
main.maps_lite(map_index).index_header.vert_offset)
vertcount_text.text = "Vertex Count: " + str(main.maps_lite(map_index).index_header.vert_count)
indsize_text.text = "Index Data Size: 0x" + _
hex(main.maps_lite(map_index).index_header.modelrawdatasize - _
main.maps_lite(map_index).index_header.ind_offset)
indcount_text.text = "Index Count: " + str(main.maps_lite(map_index).index_header.ind_count)
End Sub
main As main_Window

map_index As Integer

```

## **map\_control\_RW Control expand\_push:**

```

Sub Action()
    dim temp_map as new H1.map
    if temp_map.read(main.maps_lite(map_index).fs) then
        if temp_map.expandMap then
            main.maps_lite.Remove(map_index)
            main.maps_expanded.Append temp_map
            main.MenuBar.Child("FileMenu").Child("FileCloseMap").AutoEnable = False
            main.MenuBar.Child("MapMenu").Child("ExpandMap").AutoEnable = False
            EnableMenuItems
            main.update_list
        end
    end
End Sub
End Class

```

Class map\_rebuild\_control

Inherits ContainerControl

### **map\_rebuild\_control.Constructor:**

Sub Constructor(byref in\_w as main\_window, in\_map\_index as integer)

main = in\_w

map\_index = in\_map\_index

update

End Sub

### **map\_rebuild\_control.update:**

Sub update()

select case main.maps\_expanded(map\_index).header.version

case 609 //CE

game\_text.Text = "Game: Halo Custom Edition"

case 5 //xbox

game\_text.Text = "Game: Halo (Xbox)"

case 6 //demo

game\_text.Text = "Game: Halo Demo/Trial"

case 7 //full

game\_text.Text = "Game: Halo (PC/Mac)"

case else

game\_text.Text = "Game: Unknown"

end select

select case main.maps\_expanded(map\_index).header.maptype

case 0

maptype\_text.Text = "Map Type: Campaign"

case 1

maptype\_text.Text = "Map Type: Multiplayer"

case 2

maptype\_text.Text = "Map Type: User Interface"

case else

maptype\_text.Text = "Map Type: Unknown"

end select

name\_text.text = "Name: " + main.maps\_expanded(map\_index).header.name.replaceAll(chr(0), "")

build\_text.text = "Build: " + main.maps\_expanded(map\_index).header.builddate.replaceAll(chr(0), "")

tagcount\_text.text = "Tag Count: " + str(UBound(main.maps\_expanded(map\_index).tags) + 1)

change\_ok = false

list\_import.DeleteAllRows

list\_import.ColumnType(0) = 2

dim str\_list(-1) as string

dim ind\_list(-1) as integer

for i as integer = 0 to UBound(main.extracted\_meta\_pack)

if main.extracted\_meta\_pack(i).metas.ubound = 0 then

str\_list.append main.extracted\_meta\_pack(i).class1.reverse + ": " + main.extracted\_meta\_pack(i).name

else

str\_list.append main.extracted\_meta\_pack(i).class1.reverse + ": " + main.extracted\_meta\_pack(i).name +  
" (Recursive)"

```

    end
    ind_list.append i
next
str_list.SortWith(ind_list)
for i as integer = 0 to ind_list.Ubound
    if main.extracted_meta_pack(ind_list(i)).metas.ubound > 0 then
        list_import.AddFolder(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
            main.extracted_meta_pack(ind_list(i)).name + " (Recursive)")
        list_import.CellTag(list_import.LastIndex, 0) = "p:" + str(ind_list(i))
    else
        list_import.AddRow(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
            main.extracted_meta_pack(ind_list(i)).name)
        list_import.CellTag(list_import.LastIndex, 0) = "t:" + str(ind_list(i)) + "_" + str(0)
    end
    list_import.CellCheck(list_import.LastIndex, 0) = false
next

list_internalize.DeleteAllRows
list_internalize.ColumnType(0) = 2
list_internalize.AddFolder("Bitmaps")
list_internalize.CellTag(list_internalize.LastIndex, 0) = "b_top:-1"
list_internalize.CellCheck(list_internalize.LastIndex, 0) = false
list_internalize.AddFolder("Sounds")
list_internalize.CellTag(list_internalize.LastIndex, 0) = "s_top:-1"
list_internalize.CellCheck(list_internalize.LastIndex, 0) = false

popup_bsp.DeleteAllRows
list_bsp.DeleteAllRows
list_bsp.Hierarchical = allow_single
for i as integer = 0 to main.maps_expanded(map_index).bsp_data.ubound
    if main.maps_expanded(map_index).tagIDTable.HasKey(main.maps_expanded(map_index).bsp_data(i).tagID)
    then
        dim bsp_ind as integer = _
            main.maps_expanded(map_index).tagIDTable.value(main.maps_expanded(map_index).bsp_data(i).tagID)
        popup_bsp.AddRow("BSP " + str(i+1) + ": SBSP: " + main.maps_expanded(map_index).tags(bsp_ind).name)
    else
        break
    end
next
if main.maps_expanded(map_index).bsp_data.ubound >= 0 then
    popup_bsp.ListIndex = 0
end

redim str_list(-1)
redim ind_list(-1)
for i as integer = 0 to main.extracted_meta_pack.Ubound
    dim has_bsp as boolean = false
    dim max as integer = 0
    if allow_single then
        max = main.extracted_meta_pack(i).metas.ubound
    else
        max = 0
    end
    for j as integer = 0 to max

```

```

        if main.extracted_meta_pack(i).metas(j).class1 = "psbs" then
            has_bsp = true
            exit for j
        end
    next
    if has_bsp then
        if main.extracted_meta_pack(i).metas.ubound = 0 then
            str_list.Append main.extracted_meta_pack(i).class1.reverse + main.extracted_meta_pack(i).name
        else
            str_list.Append main.extracted_meta_pack(i).class1.reverse + main.extracted_meta_pack(i).name +
                " (Recursive)"
        end
        ind_list.Append i
    end
next

str_list.SortWith(ind_list)

for i as integer = 0 to UBound(ind_list)
    if main.extracted_meta_pack(ind_list(i)).metas.ubound = 0 then
        if allow_single then
            list_bsp.AddRow(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
                main.extracted_meta_pack(ind_list(i)).name)
            list_bsp.CellTag(list_bsp.LastIndex, 0) = "t:" + str(ind_list(i)) + "_" + str(0)
        end
    else
        if allow_single then
            list_bsp.AddFolder(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
                main.extracted_meta_pack(ind_list(i)).name + " (Recursive)")
        else
            list_bsp.AddRow(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
                main.extracted_meta_pack(ind_list(i)).name + " (Recursive)")
        end
        list_bsp.CellTag(list_bsp.LastIndex, 0) = "p:" + str(ind_list(i))
    end
next

change_ok = true
End Sub
Private allow_single As boolean

change_ok As boolean

main As main_Window

map_index As Integer

row_tags() As Integer

map_rebuild_control Control BevelButton1:

Sub Action()
    Dim file As FolderItem
    file=GetSaveFolderItem(H1_Filetypes.HaloMapFile,"build.map")

```

```

If file<> Nil then
    dim writer as new H1.map_writer
    dim temp_map as h1.map = main.maps_expanded(Map_Index)
    writer.init(temp_map, file, main, garbage_collect_check.value)
    writer.run
    main.maps_expanded(Map_Index) = temp_map
end if
End Sub

map_rebuild_control Control push_import:

Sub Action()
    if change_ok then
        dim metas(-1) as h1.meta
        dim meta_breaks(-1) as integer
        for i as integer = 0 to list_import.ListCount-1
            if list_import.CellCheck(i, 0) then //only look into it if the cell has been checked
                dim temp_str as string = list_import.CellTag(i, 0)
                dim temp_split() as string = temp_str.split(":")
                if temp_split.Ubound < 1 then return
                select case temp_split(0)
                    case "p"
                        if not list_import.Expanded(i) then // if it's expanded then don't bother, they'll be picked up later
                            meta_breaks.append metas.ubound + 1
                            for j as integer = 0 to UBound(main.extracted_meta_pack(val(temp_split(1))).metas)
                                metas.append main.extracted_meta_pack(val(temp_split(1))).metas(j)
                            next
                        end
                    case "t"
                        if main.extracted_meta_pack(val(temp_split(1))).metas.ubound = 0 then
                            meta_breaks.append metas.ubound + 1
                            metas.append main.extracted_meta_pack(val(temp_split(1))).metas(0)
                        end
                    case "t_r"
                        dim temp_split2() as string = temp_split(1).split("_")
                        if temp_split2.ubound < 1 then return
                        dim pack_index as integer = val(temp_split2(0))
                        dim meta_index as integer = val(temp_split2(1))
                        meta_breaks.append metas.ubound + 1
                        metas.append main.extracted_meta_pack(pack_index).metas(meta_index)
                    end select
                end
            next

            if metas.ubound < 0 then
                errorbox("No tags selected to import")
                return
            end if
            dim t as new map_import_thread
            t.init(main, self, metas, check_no_duplicates.Value, map_index, meta_breaks)
            t.run
        end
    End Sub

map_rebuild_control Control list_import:

```

```

Sub ExpandRow(row As Integer)
    dim temp_str as string = list_import.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split(0) <> "p" then return
    if temp_split.ubound < 1 then return
    dim index as integer = val(temp_split(1))
    dim inds(-1) as integer
    dim strings(-1) as string
    for i as integer = 0 to UBound(main.extracted_meta_pack(index).metas)
        inds.append i
        strings.append main.extracted_meta_pack(index).metas(i).class1.reverse + ": " +
            main.extracted_meta_pack(index).metas(i).name
    next
    strings.SortWith(inds)
    for i as integer = 0 to UBound(inds)
        list_import.AddRow(main.extracted_meta_pack(index).metas(inds(i)).class1.reverse + ": " +
            main.extracted_meta_pack(index).metas(inds(i)).name)
        list_import.CellTag(list_import.LastIndex, 0) = "t_r:" + str(index) + "_" + str(inds(i))
        list_import.CellCheck(list_import.LastIndex, 0) = list_import.CellCheck(row, 0)
    next
End Sub

```

```

Sub CellAction(row As Integer, column As Integer)
    if not change_ok then return
    dim temp_str as string = list_import.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.Ubound < 1 then return
    select case temp_split(0)
    case "p"
        //folder, check to see if it's open
        if list_import.Expanded(row) then
            //it's expanded, go through all of the children
            dim index as integer = val(temp_split(1))
            for i as integer = 0 to UBound(main.extracted_meta_pack(index).metas)
                list_import.CellCheck(row+1+i,0) = list_import.CellCheck(row,0)
            next
        end
    case "t"
        //do nothing
    case "t_r"
        //it's an item in a pack list
        dim temp_split2() as string = temp_split(1).split("_")
        dim pack_index as integer = val(temp_split2(0))
        dim parent_row as integer = -1
        for i as integer = 0 to list_import.ListCount - 1
            dim temp_str_new as string = list_import.CellTag(i,0)
            dim temp_split_new() as string = temp_str_new.split(":")
            if temp_split_new.Ubound < 1 then return
            if temp_split_new(0) = "p" and val(temp_split_new(1)) = pack_index then
                parent_row = i
                exit for i
            end
        next
        if parent_row < 0 then return
    end select

```

```

if list_import.CellCheck(row, 0) then
    //it was checked
    //need to check if everything else is enabled
    dim all_enabled as boolean = true
    for i as integer = 0 to UBound(main.extracted_meta_pack(pack_index).metas)
        if list_import.CellCheck(parent_row + 1 + i, 0) = false then
            //found an unchecked box in the group so do nothing
            all_enabled = false
            exit for i
        end
    next
    if all_enabled then
        //all of the children were enabled, so enable to parent
        change_ok = false
        list_import.CellCheck(parent_row, 0) = true
        change_ok = true
    end
else
    //it was unchecked
    //this is simple, just need to uncheck the parent row
    change_ok = false
    list_import.CellCheck(parent_row, 0) = false
    change_ok = true
end
end select
End Sub

```

### **map\_rebuild\_control Control list\_internalize:**

```

Sub ExpandRow(row As Integer)
    dim temp_str as string = list_internalize.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then return
    dim type_str as string = temp_split(0)

    select case type_str

    case "b_top"
        dim bitm_folder_index as integer = -1
        dim bitm_inds(-1) as integer
        for i as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
            if main.maps_expanded(Map_Index).meta_class_folders.child(i).name.mid(1,4) = "bitm" then
                bitm_folder_index = i
                exit for i
            end
        next
        if bitm_folder_index = -1 then return
        for i as integer = 0 to
            UBound(main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item)
            dim tag_index as integer =
                main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item(i)
            dim magic as integer = main.maps_expanded(Map_index).magic
            if main.maps_expanded(Map_index).tags(tag_index).magic <> -1 then
                magic = main.maps_expanded(Map_index).tags(tag_index).magic
            end
        next
    end

```

```

    dim b as new bitmap_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
    main.maps_expanded(Map_index).tags(tag_index).offset, magic)
    b.read
    dim has_external as boolean = false
    for j as integer = 0 to UBound(b.image_info)
        if (bitand(b.image_info(j).flags, &h100) = &h100) then
            has_external = true
            exit for j
        end
    next
    if has_external then
        bitm_inds.append tag_index
    else
        dim temp as integer
    end
next

dim string_list(-1) as string
for i as integer = 0 to UBound(bitm_inds)
    string_list.append main.maps_expanded(Map_Index).tags(bitm_inds(i)).name
next
string_list.sortwith(bitm_inds)

for i as integer = 0 to UBound(bitm_inds)
    dim tag_index as integer = bitm_inds(i)
    list_internalize.AddRow(main.maps_expanded(Map_Index).tags(tag_index).class1.reverse + ": " +
    main.maps_expanded(Map_Index).tags(tag_index).name)
    list_internalize.CellTag(list_internalize.LastIndex, 0) = "b:" + str(tag_index)
    list_internalize.CellCheck(list_internalize.LastIndex, 0) = list_internalize.CellCheck(row, 0)
next

case "s_top"
    dim snd_folder_index as integer = -1
    dim snd_inds(-1) as integer
    for i as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
        if main.maps_expanded(Map_Index).meta_class_folders.child(i).name.mid(1,4) = "snd!" then
            snd_folder_index = i
            exit for i
        end
    next
    if snd_folder_index = -1 then return
    for i as integer = 0 to
    UBound(main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item)
        dim tag_index as integer =
        main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item(i)
        dim magic as integer = main.maps_expanded(Map_index).magic
        if main.maps_expanded(Map_index).tags(tag_index).magic <> -1 then
            magic = main.maps_expanded(Map_index).tags(tag_index).magic
        end
        dim s as new snd_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
        main.maps_expanded(Map_index).tags(tag_index).offset, magic)
        s.read
        dim has_external as boolean = false
        for j as integer = 0 to UBound(s.pitch)

```



```

        for k as integer = 0 to UBound(s.pitch(j).permutations)
            if not s.pitch(j).permutations(k).internal then
                has_external = true
                exit for j
            end
        next
    next
    if has_external then
        snd_inds.append tag_index
    end
next

dim string_list(-1) as string
for i as integer = 0 to UBound(snd_inds)
    string_list.append main.maps_expanded(Map_Index).tags(snd_inds(i)).name
next
string_list.sortwith(snd_inds)

for i as integer = 0 to UBound(snd_inds)
    dim tag_index as integer = snd_inds(i)
    list_internalize.AddRow(main.maps_expanded(Map_Index).tags(tag_index).class1.reverse + ": " +
        main.maps_expanded(Map_Index).tags(tag_index).name)
    list_internalize.CellTag(list_internalize.LastIndex, 0) = "s:" + str(tag_index)
    list_internalize.CellCheck(list_internalize.LastIndex, 0) = list_internalize.CellCheck(row, 0)
next

case "b"
    //should not fire

case "s"
    //should not fire

end select
End Sub

Sub CellAction(row As Integer, column As Integer)
    if not change_ok then return
    dim temp_str as string = list_internalize.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then return
    dim type_str as string = temp_split(0)

    select case type_str

    case "b_top"
        //alter all children
        if list_internalize.Expanded(row) then
            for i as integer = 0 to list_internalize.ListCount - 1
                dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
                if temp_split2.Ubound < 1 then continue
                dim type_str2 as string = temp_split2(0)
                if type_str2 = "b" then
                    list_internalize.CellCheck(i, 0) = list_internalize.CellCheck(row,0)
                end
            next
        end
    end select
end Sub

```

```

    next
end

case "s_top"
    //alter all children
    if list_internalize.Expanded(row) then
        for i as integer = 0 to list_internalize.ListCount - 1
            dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
            if temp_split2.Ubound < 1 then continue
            dim type_str2 as string = temp_split2(0)
            if type_str2 = "s" then
                list_internalize.CellCheck(i, 0) = list_internalize.CellCheck(row,0)
            end
        next
    end
next
end

case "b"
    dim parent_row as integer = -1
    for i as integer = 0 to list_internalize.ListCount - 1
        dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
        if temp_split2.Ubound < 1 then continue
        dim type_str2 as string = temp_split2(0)
        if type_str2 = "b_top" then
            parent_row = i
            exit for i
        end
    next
    if parent_row < 0 then return
    if list_internalize.CellCheck(row, 0) then
        dim all_enabled as boolean = true
        //search through all children, if all are true then parent is true otherwise do nothing
        for i as integer = 0 to list_internalize.ListCount - 1
            dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
            if temp_split2.Ubound < 1 then continue
            dim type_str2 as string = temp_split2(0)
            if type_str2 = "b" then
                if not list_internalize.CellCheck(i, 0) then
                    all_enabled = false
                    exit for i
                end
            end
        next
        if all_enabled then
            change_ok = false
            list_internalize.CellCheck(parent_row, 0) = true
            change_ok = true
        end
    else
        //unchecked simply uncheck the parent
        change_ok = false
        list_internalize.CellCheck(parent_row, 0) = false
        change_ok = true
    end
end

```

```

case "s"
    dim parent_row as integer = -1
    for i as integer = 0 to list_internalize.ListCount - 1
        dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
        if temp_split2.Ubound < 1 then continue
        dim type_str2 as string = temp_split2(0)
        if type_str2 = "s_top" then
            parent_row = i
            exit for i
        end
    next
    if parent_row < 0 then return
    if list_internalize.CellCheck(row, 0) then
        dim all_enabled as boolean = true
        //search through all children, if all are true then parent is true otherwise do nothing
        for i as integer = 0 to list_internalize.ListCount - 1
            dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
            if temp_split2.Ubound < 1 then continue
            dim type_str2 as string = temp_split2(0)
            if type_str2 = "s" then
                if not list_internalize.CellCheck(i, 0) then
                    all_enabled = false
                    exit for i
                end
            end
        next
        if all_enabled then
            change_ok = false
            list_internalize.CellCheck(parent_row, 0) = true
            change_ok = true
        end
    else
        //unchecked simply uncheck the parent
        change_ok = false
        list_internalize.CellCheck(parent_row, 0) = false
        change_ok = true
    end
end

```

end select

End Sub

### **map\_rebuild\_control Control push\_internalize:**

Sub Action()

```

dim bitmap_indexs(-1) as integer
dim sound_indexs(-1) as integer

for i as integer = 0 to list_internalize.ListCount - 1
    dim temp_str as string = list_internalize.CellTag(i, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then continue
    dim type_str as string = temp_split(0)
    if list_internalize.CellCheck(i, 0) then

```

```

select case type_str
case "b_top"
    //if it's expanded do nothing and let the other parts take care of finding items
    if not list_internalize.Expanded(i) then
        //go through the check for non-internalized bitmaps

        dim bitm_folder_index as integer = -1
        for j as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
            if main.maps_expanded(Map_Index).meta_class_folders.child(j).name.mid(1,4) = "bitm" then
                bitm_folder_index = j
                exit for j
            end
        next
        if bitm_folder_index = -1 then continue
        for j as integer = 0 to
            UBound(main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item)
            dim tag_index as integer =
                main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item(j)
            dim b as new bitmap_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
                main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)
            b.read
            dim has_external as boolean = false
            for k as integer = 0 to UBound(b.image_info)
                if (bitand(b.image_info(k).flags, &h100) = &h100) then
                    has_external = true
                    exit for k
                end
            next
            if has_external then
                bitmap_indexes.Append tag_index
            end
        next
    end

case "s_top"
    //if it's expanded do nothing and let the other parts take care of finding items
    if not list_internalize.Expanded(i) then
        //go through the check for non-internalized sounds

        dim snd_folder_index as integer = -1
        for j as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
            if main.maps_expanded(Map_Index).meta_class_folders.child(j).name.mid(1,4) = "snd!" then
                snd_folder_index = j
                exit for j
            end
        next
        if snd_folder_index = -1 then return
        for j as integer = 0 to
            UBound(main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item)
            dim tag_index as integer =
                main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item(j)
            dim s as new snd_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
                main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)

```

```

        s.read
        dim has_external as boolean = false
        for k as integer = 0 to UBound(s.pitch)
            for l as integer = 0 to UBound(s.pitch(k).permutations)
                if not s.pitch(k).permutations(l).internal then
                    has_external = true
                    exit for k
                end
            next
        next
        if has_external then
            sound_indexes.Append tag_index
        end
    next

end

case "b"
    bitmap_indexes.Append val(temp_split(1))

case "s"
    sound_indexes.Append val(temp_split(1))

end select
end
next

Dim bitmaps_f as FolderItem
if bitmap_indexes.ubound > -1 then
    if not change_ok then return
    Dim dlg as OpenFileDialog
    dlg= New OpenFileDialog
    dlg.Title="Select a Bitmaps.map file"
    dlg.Filter=H1_Filetypes.HaloMapFile
    bitmaps_f=dlg.ShowModal()
    If bitmaps_f = Nil then return
    if not bitmaps_f.exists then return
else
    bitmaps_f = nil
end

Dim sounds_f as FolderItem
if sound_indexes.ubound > -1 then
    Dim dlg2 as OpenFileDialog
    dlg2= New OpenFileDialog
    dlg2.Title="Select a Sounds.map file"
    dlg2.Filter=H1_Filetypes.HaloMapFile
    sounds_f=dlg2.ShowModal()
    If sounds_f = Nil then return
    if not sounds_f.exists then return
else
    sounds_f = nil
end

```

```

if bitmap_indexs.ubound < 0 and sound_indexs.ubound < 0 then
    errorbox("No tags selected to internalize")
    return
end if

dim t as new map_internalize_thread
t.init(self, main, map_index, bitmaps_f, sounds_f, bitmap_indexs, sound_indexs)
t.run

```

End Sub

### **map\_rebuild\_control Control list\_bsp:**

```

Sub ExpandRow(row As Integer)
    if not change_ok then return
    dim temp_split() as string = me.CellTag(row, 0).split(":")
    if temp_split.Ubound < 1 then return
    select case temp_split(0)
    case "m"
        //should never fire
    case "p"
        //meta pack, expand it
        dim index as integer = val(temp_split(1))
        dim inds(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(main.extracted_meta_pack(index).metas)
            if main.extracted_meta_pack(index).metas(i).class1 = "psbs" then //only include sbsp tags
                inds.append i
                strings.append main.extracted_meta_pack(index).metas(i).class1.reverse + ": " +
                    main.extracted_meta_pack(index).metas(i).name
            end
        next
        strings.SortWith(inds)
        for i as integer = 0 to UBound(inds)
            me.AddRow(main.extracted_meta_pack(index).metas(inds(i)).class1.reverse + ": " +
                main.extracted_meta_pack(index).metas(inds(i)).name)
            me.CellTag(me.LastIndex, 0) = "t:" + str(index) + "_" + str(inds(i))
        next
    case else
        //should never fire
    end select
End Sub

```

### **map\_rebuild\_control Control push\_bsp:**

```

Sub Action()
    dim index as integer = list_bsp.ListIndex
    if index = -1 then
        errorbox("Error: No BSP selected")
        return
    end
    dim temp_str as string = list_bsp.CellTag(index, 0)

    //t:packind_tagind
    //p:packind

    dim bsp_data_index as integer = popup_bsp.ListIndex

```

```

if bsp_data_index = -1 then return
dim replacementindex as integer
if
main.maps_expanded(map_index).tagIDTable.haskey(main.maps_expanded(map_index).bsp_data(bsp_data_index).tagID) then
    replacementindex = _
    main.maps_expanded(map_index).tagIDTable.value(main.maps_expanded(map_index).bsp_data(bsp_data_index).tagID)
else
    return
end

```

```

dim temp_strs() as string = temp_str.Split(":")
if temp_strs.Ubound <> 1 then return //error
select case temp_strs(0)
case "t"
    //not going to allow single SBSP replacement
    //main.maps_expanded(map_index).replaceSBSP(replacementindex, main.
case "p"
    //this needs a thread
    dim t as new map_BSP_replace_thread
    t.init(main, self, map_index, val(temp_strs(1)), replacementindex, check_no_duplicates_bsp.Value)
    t.run
end select

```

End Sub

## **map\_rebuild\_control Control BevelButton2:**

Sub Action()

```

dim class_index as integer = -1
for i as integer = 0 to UBound(main.maps_expanded(map_index).meta_class_folders.child)
    if main.maps_expanded(map_index).meta_class_folders.child(i).name.mid(1,4) = "mod2" then
        class_index = i
        exit for i
    end
next
if class_index = -1 then return

dim diff_sized_tags(-1) as integer
dim diff_sized_tags_other(-1) as integer
for i as integer = 0 to UBound(main.maps_expanded(map_index).tags)
    dim temp_size as integer = main.maps_expanded(map_index).tags(i).metasize
    dim temp_key as string = main.maps_expanded(map_index).tags(i).class1 + _
    ":" + main.maps_expanded(map_index).tags(i).name
    for j as integer = 0 to main.maps_expanded.Ubound
        if j <> Map_Index then
            if main.maps_expanded(j).tagClassNameTable.haskey(temp_key) then
                dim temp_ind as integer = main.maps_expanded(j).tagClassNameTable.value(temp_key)
                if main.maps_expanded(j).tags(temp_ind).metasize <> temp_size then
                    diff_sized_tags.Append(i)
                    diff_sized_tags_other.Append(temp_ind)
                end
            end
        end
    end
next

```

```

next
if diff_sized_tags.Ubound > -1 then
    dim breaker as integer
end
'dim vert_total as integer = 0
'dim ind_total as integer = 0
'dim raw_total as integer = 0
'for i as integer = 0 to UBound(main.maps_expanded(map_index).meta_class_folders.child(class_index).item)
'dim tag_ind as integer = main.maps_expanded(map_index).meta_class_folders.child(class_index).item(i)
'dim vert_count as integer = 0
'dim ind_count as integer = 0
'dim raw_count as integer = 0
'dim tag_str as string = main.maps_expanded(map_index).tags(tag_ind).class1 + ":" + _
'main.maps_expanded(map_index).tags(tag_ind).name
'for j as integer = 0 to main.maps_expanded(map_index).tags(tag_ind).raw_data.ubound
'dim temp_str() as string = split(main.maps_expanded(map_index).tags(tag_ind).raw_data(j).str_info, ":")
'if temp_str(UBound(temp_str)) = "vert" then
'vert_count = vert_count + 1
'end
'if temp_str(UBound(temp_str)) = "ind" then
'ind_count = ind_count + 1
'end
'raw_count = raw_count + 1
'next
,

'if raw_count <> main.maps_expanded(map_index).tags(tag_ind).raw_data.ubound + 1 then
'dim break_raw as integer
'end
,

'vert_total = vert_total + vert_count
'ind_total = ind_total + ind_count
'raw_total = raw_total + raw_count
,

'for j as integer = 0 to UBound(main.maps_expanded)
'if j <> map_index then
'if main.maps_expanded(j).tagClassNameTable.haskey(tag_str) then
'dim new_ind as integer = main.maps_expanded(j).tagClassNameTable.value(tag_str)
'dim new_vert_count as integer = 0
'dim new_ind_count as integer = 0
'for k as integer = 0 to main.maps_expanded(j).tags(new_ind).raw_data.ubound
'dim temp_str() as string = split(main.maps_expanded(j).tags(new_ind).raw_data(k).str_info, ":")
'if temp_str(UBound(temp_str)) = "vert" then
'new_vert_count = new_vert_count + 1
'end
'if temp_str(UBound(temp_str)) = "ind" then
'new_ind_count = new_ind_count + 1
'end
'next
'if new_vert_count <> vert_count then
'dim break_verts as integer
'end
'if new_ind_count <> ind_count then
'dim break_ind as integer
'end

```



```

'end
'end
'next
'next
'
'dim end_point as integer
End Sub
End Class

```

## **Class map\_rebuild\_control\_old**

Inherits ContainerControl

### **map\_rebuild\_control\_old.Constructor:**

```

Sub Constructor(byref in_w as main_window, in_map_index as integer)
    main = in_w
    map_index = in_map_index

    update
End Sub

```

### **map\_rebuild\_control\_old.update:**

```

Sub update()
    select case main.maps_expanded(map_index).header.version
    case 609 //CE
        game_text.Text = "Game: Halo Custom Edition"
    case 5 //xbox
        game_text.Text = "Game: Halo (Xbox)"
    case 6 //demo
        game_text.Text = "Game: Halo Demo/Trial"
    case 7 //full
        game_text.Text = "Game: Halo (PC/Mac)"
    case else
        game_text.Text = "Game: Unknown"
    end select

    select case main.maps_expanded(map_index).header.maptype
    case 0
        maptype_text.Text = "Map Type: Campaign"
    case 1
        maptype_text.Text = "Map Type: Multiplayer"
    case 2
        maptype_text.Text = "Map Type: User Interface"
    case else
        maptype_text.Text = "Map Type: Unknown"
    end select

    name_text.text = "Name: " + main.maps_expanded(map_index).header.name.replaceAll(chr(0), "")
    build_text.text = "Build: " + main.maps_expanded(map_index).header.builddate.replaceAll(chr(0), "")
    tagcount_text.text = "Tag Count: " + str(UBound(main.maps_expanded(map_index).tags) + 1)

    change_ok = false
    list_import.DeleteAllRows

```

```

list_import.ColumnType(0) = 2
dim str_list(-1) as string
dim ind_list(-1) as integer
for i as integer = 0 to UBound(main.extracted_meta_pack)
    if main.extracted_meta_pack(i).metas.ubound = 0 then
        str_list.append main.extracted_meta_pack(i).class1.reverse + ": " + main.extracted_meta_pack(i).name
    else
        str_list.append main.extracted_meta_pack(i).class1.reverse + ": " + main.extracted_meta_pack(i).name +
            " (Recursive)"
    end
    ind_list.append i
next
str_list.SortWith(ind_list)
for i as integer = 0 to ind_list.Ubound
    if main.extracted_meta_pack(ind_list(i)).metas.ubound > 0 then
        list_import.AddFolder(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
            main.extracted_meta_pack(ind_list(i)).name + " (Recursive)")
        list_import.CellTag(list_import.LastIndex, 0) = "p:" + str(ind_list(i))
    else
        list_import.AddRow(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
            main.extracted_meta_pack(ind_list(i)).name)
        list_import.CellTag(list_import.LastIndex, 0) = "t:" + str(ind_list(i)) + "_" + str(0)
    end
    list_import.CellCheck(list_import.LastIndex, 0) = false
next

list_internalize.DeleteAllRows
list_internalize.ColumnType(0) = 2
list_internalize.AddFolder("Bitmaps")
list_internalize.CellTag(list_internalize.LastIndex, 0) = "b_top:-1"
list_internalize.CellCheck(list_internalize.LastIndex, 0) = false
list_internalize.AddFolder("Sounds")
list_internalize.CellTag(list_internalize.LastIndex, 0) = "s_top:-1"
list_internalize.CellCheck(list_internalize.LastIndex, 0) = false
change_ok = true
End Sub
change_ok As boolean

main As main_Window

map_index As Integer

row_tags() As Integer

```

### **map\_rebuild\_control\_old Control BevelButton1:**

```

Sub Action()
    Dim file As FolderItem
    file=GetSaveFolderItem(H1.Filetypes.HaloMapFile,"build.map")
    If file<> Nil then
        dim writer as new H1.map_writer
        dim temp_map as h1.map = main.maps_expanded(Map_Index)
        writer.init(temp_map, file, main)
        writer.run
    end if
end sub

```

```

    main.maps_expanded(Map_Index) = temp_map
end if
End Sub

```

### **map\_rebuild\_control\_old Control push\_import:**

```

Sub Action()
    if change_ok then
        dim metas(-1) as h1.meta
        for i as integer = 0 to list_import.ListCount-1
            if list_import.CellCheck(i, 0) then //only look into it if the cell has been checked
                dim temp_str as string = list_import.CellTag(i, 0)
                dim temp_split() as string = temp_str.split(":")
                if temp_split.Ubound < 1 then return
                select case temp_split(0)
                    case "p"
                        if not list_import.Expanded(i) then // if it's expanded then don't bother, they'll be picked up later
                            for j as integer = 0 to UBound(main.extracted_meta_pack(val(temp_split(1))).metas)
                                metas.append main.extracted_meta_pack(val(temp_split(1))).metas(j)
                            next
                        end
                    case "t"
                        if main.extracted_meta_pack(val(temp_split(1))).metas.ubound = 0 then
                            metas.append main.extracted_meta_pack(val(temp_split(1))).metas(0)
                        end
                    case "t_r"
                        dim temp_split2() as string = temp_split(1).split("_")
                        if temp_split2.ubound < 1 then return
                        dim pack_index as integer = val(temp_split2(0))
                        dim meta_index as integer = val(temp_split2(1))
                        metas.append main.extracted_meta_pack(pack_index).metas(meta_index)
                    end select
                end
            next

            if metas.ubound < 0 then
                errorbox("No tags selected to import")
                return
            end if
            BREAK //need to fix this heya if you want to reimplement
            //dim t as new map_import_thread
            //t.init(main, self, metas, check_no_duplicates.Value, map_index)
            //t.run
        end
    End Sub

```

### **map\_rebuild\_control\_old Control list\_import:**

```

Sub ExpandRow(row As Integer)
    dim temp_str as string = list_import.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split(0) <> "p" then return
    if temp_split.ubound < 1 then return
    dim index as integer = val(temp_split(1))
    dim inds(-1) as integer
    dim strings(-1) as string

```

```

for i as integer = 0 to UBound(main.extracted_meta_pack(index).metas)
    inds.append i
    strings.append main.extracted_meta_pack(index).metas(i).class1.reverse + ": " +
        main.extracted_meta_pack(index).metas(i).name
next
strings.SortWith(inds)
for i as integer = 0 to UBound(inds)
    list_import.AddRow(main.extracted_meta_pack(index).metas(inds(i)).class1.reverse + ": " +
        main.extracted_meta_pack(index).metas(inds(i)).name)
    list_import.CellTag(list_import.LastIndex, 0) = "t_r:" + str(index) + "_" + str(inds(i))
    list_import.CellCheck(list_import.LastIndex, 0) = list_import.CellCheck(row, 0)
next
End Sub

```

```

Sub CellAction(row As Integer, column As Integer)
    if not change_ok then return
    dim temp_str as string = list_import.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.Ubound < 1 then return
    select case temp_split(0)
    case "p"
        //folder, check to see if it's open
        if list_import.Expanded(row) then
            //it's expanded, go through all of the children
            dim index as integer = val(temp_split(1))
            for i as integer = 0 to UBound(main.extracted_meta_pack(index).metas)
                list_import.CellCheck(row+1+i,0) = list_import.CellCheck(row,0)
            next
        end
    case "t"
        //do nothing
    case "t_r"
        //it's an item in a pack list
        dim temp_split2() as string = temp_split(1).split("_")
        dim pack_index as integer = val(temp_split2(0))
        dim parent_row as integer = -1
        for i as integer = 0 to list_import.ListCount - 1
            dim temp_str_new as string = list_import.CellTag(i,0)
            dim temp_split_new() as string = temp_str_new.split(":")
            if temp_split_new.Ubound < 1 then return
            if temp_split_new(0) = "p" and val(temp_split_new(1)) = pack_index then
                parent_row = i
                exit for i
            end
        next
        if parent_row < 0 then return
        if list_import.CellCheck(row, 0) then
            //it was checked
            //need to check if everything else is enabled
            dim all_enabled as boolean = true
            for i as integer = 0 to UBound(main.extracted_meta_pack(pack_index).metas)
                if list_import.CellCheck(parent_row + 1 + i, 0) = false then
                    //found an unchecked box in the group so do nothing
                    all_enabled = false
                end if
            next
        end if
    end select
End Sub

```

```

        exit for i
    end
next
if all_enabled then
    //all of the children were enabled, so enable to parent
    change_ok = false
    list_import.CellCheck(parent_row, 0) = true
    change_ok = true
end
else
    //it was unchecked
    //this is simple, just need to uncheck the parent row
    change_ok = false
    list_import.CellCheck(parent_row, 0) = false
    change_ok = true
end
end select
End Sub

```

### **map\_rebuild\_control\_old Control list\_internalize:**

```

Sub ExpandRow(row As Integer)
    dim temp_str as string = list_internalize.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then return
    dim type_str as string = temp_split(0)

    select case type_str

    case "b_top"
        dim bitm_folder_index as integer = -1
        dim bitm_inds(-1) as integer
        for i as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
            if main.maps_expanded(Map_Index).meta_class_folders.child(i).name.mid(1,4) = "bitm" then
                bitm_folder_index = i
                exit for i
            end
        next
        if bitm_folder_index = -1 then return
        for i as integer = 0 to
            UBound(main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item)
            dim tag_index as integer =
                main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item(i)
            dim b as new bitmap_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
                main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)
            b.read
            dim has_external as boolean = false
            for j as integer = 0 to UBound(b.image_info)
                if (bitand(b.image_info(j).flags, &h100) = &h100) then
                    has_external = true
                    exit for j
                end
            next
            if has_external then
                bitm_inds.append tag_index
            end
        next
    end select
End Sub

```

```

    end
next

dim string_list(-1) as string
for i as integer = 0 to UBound(bitm_inds)
    string_list.append main.maps_expanded(Map_Index).tags(bitm_inds(i)).name
next
string_list.sortwith(bitm_inds)

for i as integer = 0 to UBound(bitm_inds)
    dim tag_index as integer = bitm_inds(i)
    list_internalize.AddRow(main.maps_expanded(Map_Index).tags(tag_index).class1.reverse + ": " +
        main.maps_expanded(Map_Index).tags(tag_index).name)
    list_internalize.CellTag(list_internalize.LastIndex, 0) = "b:" + str(tag_index)
    list_internalize.CellCheck(list_internalize.LastIndex, 0) = list_internalize.CellCheck(row, 0)
next

case "s_top"
    dim snd_folder_index as integer = -1
    dim snd_inds(-1) as integer
    for i as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
        if main.maps_expanded(Map_Index).meta_class_folders.child(i).name.mid(1,4) = "snd!" then
            snd_folder_index = i
            exit for i
        end
    next
    if snd_folder_index = -1 then return
    for i as integer = 0 to
        UBound(main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item)
        dim tag_index as integer =
            main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item(i)
        dim s as new snd_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
            main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)
        s.read
        dim has_external as boolean = false
        for j as integer = 0 to UBound(s.pitch)
            for k as integer = 0 to UBound(s.pitch(j).permutations)
                if not s.pitch(j).permutations(k).internal then
                    has_external = true
                    exit for j
                end
            next
        next
        if has_external then
            snd_inds.append tag_index
        end
    next

    dim string_list(-1) as string
    for i as integer = 0 to UBound(snd_inds)
        string_list.append main.maps_expanded(Map_Index).tags(snd_inds(i)).name
    next
    string_list.sortwith(snd_inds)

```

```

for i as integer = 0 to UBound(snd_inds)
    dim tag_index as integer = snd_inds(i)
    list_internalize.AddRow(main.maps_expanded(Map_Index).tags(tag_index).class1.reverse + ": " +
        main.maps_expanded(Map_Index).tags(tag_index).name)
    list_internalize.CellTag(list_internalize.LastIndex, 0) = "s:" + str(tag_index)
    list_internalize.CellCheck(list_internalize.LastIndex, 0) = list_internalize.CellCheck(row, 0)
next

case "b"
    //should not fire

case "s"
    //should not fire

end select
End Sub

Sub CellAction(row As Integer, column As Integer)
    if not change_ok then return
    dim temp_str as string = list_internalize.CellTag(row, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then return
    dim type_str as string = temp_split(0)

    select case type_str

    case "b_top"
        //alter all children
        if list_internalize.Expanded(row) then
            for i as integer = 0 to list_internalize.ListCount - 1
                dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
                if temp_split2.Ubound < 1 then continue
                dim type_str2 as string = temp_split2(0)
                if type_str2 = "b" then
                    list_internalize.CellCheck(i, 0) = list_internalize.CellCheck(row,0)
                end
            next
        end

    case "s_top"
        //alter all children
        if list_internalize.Expanded(row) then
            for i as integer = 0 to list_internalize.ListCount - 1
                dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
                if temp_split2.Ubound < 1 then continue
                dim type_str2 as string = temp_split2(0)
                if type_str2 = "s" then
                    list_internalize.CellCheck(i, 0) = list_internalize.CellCheck(row,0)
                end
            next
        end

    case "b"
        dim parent_row as integer = -1

```

```

for i as integer = 0 to list_internalize.ListCount - 1
    dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
    if temp_split2.Ubound < 1 then continue
    dim type_str2 as string = temp_split2(0)
    if type_str2 = "b_top" then
        parent_row = i
        exit for i
    end
next
if parent_row < 0 then return
if list_internalize.CellCheck(row, 0) then
    dim all_enabled as boolean = true
    //search through all children, if all are true then parent is true otherwise do nothing
    for i as integer = 0 to list_internalize.ListCount - 1
        dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
        if temp_split2.Ubound < 1 then continue
        dim type_str2 as string = temp_split2(0)
        if type_str2 = "b" then
            if not list_internalize.CellCheck(i, 0) then
                all_enabled = false
                exit for i
            end
        end
    next
    if all_enabled then
        change_ok = false
        list_internalize.CellCheck(parent_row, 0) = true
        change_ok = true
    end
else
    //unchecked simply uncheck the parent
    change_ok = false
    list_internalize.CellCheck(parent_row, 0) = false
    change_ok = true
end

case "s"
dim parent_row as integer = -1
for i as integer = 0 to list_internalize.ListCount - 1
    dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
    if temp_split2.Ubound < 1 then continue
    dim type_str2 as string = temp_split2(0)
    if type_str2 = "s_top" then
        parent_row = i
        exit for i
    end
next
if parent_row < 0 then return
if list_internalize.CellCheck(row, 0) then
    dim all_enabled as boolean = true
    //search through all children, if all are true then parent is true otherwise do nothing
    for i as integer = 0 to list_internalize.ListCount - 1
        dim temp_split2() as string = list_internalize.CellTag(i,0).split(":")
        if temp_split2.Ubound < 1 then continue
    next

```



```

    dim type_str2 as string = temp_split2(0)
    if type_str2 = "s" then
        if not list_internalize.CellCheck(i, 0) then
            all_enabled = false
            exit for i
        end
    end
next
if all_enabled then
    change_ok = false
    list_internalize.CellCheck(parent_row, 0) = true
    change_ok = true
end
else
    //unchecked simply uncheck the parent
    change_ok = false
    list_internalize.CellCheck(parent_row, 0) = false
    change_ok = true
end

```

```

end select
End Sub

```

### **map\_rebuild\_control\_old Control push\_internalize:**

```

Sub Action()

```

```

    dim bitmap_indexs(-1) as integer
    dim sound_indexs(-1) as integer

```

```

for i as integer = 0 to list_internalize.ListCount - 1
    dim temp_str as string = list_internalize.CellTag(i, 0)
    dim temp_split() as string = temp_str.split(":")
    if temp_split.ubound < 1 then continue
    dim type_str as string = temp_split(0)
    if list_internalize.CellCheck(i, 0) then
        select case type_str
            case "b_top"
                //if it's expanded do nothing and let the other parts take care of finding items
                if not list_internalize.Expanded(i) then
                    //go through the check for non-internalized bitmaps

                    dim bitm_folder_index as integer = -1
                    for j as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
                        if main.maps_expanded(Map_Index).meta_class_folders.child(j).name.mid(1,4) = "bitm" then
                            bitm_folder_index = j
                            exit for j
                        end
                    next
                    if bitm_folder_index = -1 then continue
                    for j as integer = 0 to
                        UBound(main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item)
                        dim tag_index as integer =
                            main.maps_expanded(Map_Index).meta_class_folders.child(bitm_folder_index).item(j)

```

```

    dim b as new bitmap_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
    main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)
    b.read
    dim has_external as boolean = false
    for k as integer = 0 to UBound(b.image_info)
        if (bitand(b.image_info(k).flags, &h100) = &h100) then
            has_external = true
            exit for k
        end
    next
    if has_external then
        bitmap_indexes.Append tag_index
    end
next

end

case "s_top"
    //if it's expanded do nothing and let the other parts take care of finding items
    if not list_internalize.Expanded(i) then
        //go through the check for non-internalized sounds

        dim snd_folder_index as integer = -1
        for j as integer = 0 to UBound(main.maps_expanded(Map_Index).meta_class_folders.child)
            if main.maps_expanded(Map_Index).meta_class_folders.child(j).name.mid(1,4) = "snd!" then
                snd_folder_index = j
                exit for j
            end
        next
        if snd_folder_index = -1 then return
        for j as integer = 0 to
            UBound(main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item)
            dim tag_index as integer =
                main.maps_expanded(Map_Index).meta_class_folders.child(snd_folder_index).item(j)
            dim s as new snd_class_EX(main.maps_expanded(Map_index).tags(tag_index).data.bin, _
                main.maps_expanded(Map_index).tags(tag_index).offset, main.maps_expanded(Map_index).magic)
            s.read
            dim has_external as boolean = false
            for k as integer = 0 to UBound(s.pitch)
                for l as integer = 0 to UBound(s.pitch(k).permutations)
                    if not s.pitch(k).permutations(l).internal then
                        has_external = true
                        exit for k
                    end
                next
            next
            if has_external then
                sound_indexes.Append tag_index
            end
        next

    end

end

case "b"

```

```

        bitmap_indexs.Append val(temp_split(1))

    case "s"
        sound_indexs.Append val(temp_split(1))

    end select
end
next

Dim bitmaps_f as FolderItem
if bitmap_indexs.ubound > -1 then
    if not change_ok then return
    Dim dlg as OpenFileDialog
    dlg= New OpenFileDialog
    dlg.Title="Select a Bitmaps.map file"
    dlg.Filter=H1_Filetypes.HaloMapFile
    bitmaps_f=dlg.ShowModal()
    If bitmaps_f = Nil then return
    if not bitmaps_f.exists then return
else
    bitmaps_f = nil
end

Dim sounds_f as FolderItem
if sound_indexs.ubound > -1 then
    Dim dlg2 as OpenFileDialog
    dlg2= New OpenFileDialog
    dlg2.Title="Select a Sounds.map file"
    dlg2.Filter=H1_Filetypes.HaloMapFile
    sounds_f=dlg2.ShowModal()
    If sounds_f = Nil then return
    if not sounds_f.exists then return
else
    sounds_f = nil
end

if bitmap_indexs.ubound < 0 and sound_indexs.ubound < 0 then
    errorbox("No tags selected to internalize")
    return
end if

BREAK //need to fix this heya if you want to reimplement
//dim t as new map_internalize_thread
//t.init(self, main, map_index, bitmaps_f, sounds_f, bitmap_indexs, sound_indexs)
//t.run
End Sub
End Class

```

## **Class map\_rebuild\_control\_old\_old**

Inherits ContainerControl

**map\_rebuild\_control\_old\_old.Constructor:**

```
Sub Constructor(byref in_w as main_window, in_map_index as integer)
```

```
    main = in_w
```

```
    map_index = in_map_index
```

```
    update
```

```
End Sub
```

### **map\_rebuild\_control\_old\_old.update:**

```
Sub update()
```

```
    select case main.maps_expanded(map_index).header.version
```

```
    case 609 //CE
```

```
        game_text.Text = "Game: Halo Custom Edition"
```

```
    case 5 //xbox
```

```
        game_text.Text = "Game: Halo (Xbox)"
```

```
    case 6 //demo
```

```
        game_text.Text = "Game: Halo Demo/Trial"
```

```
    case 7 //full
```

```
        game_text.Text = "Game: Halo (PC/Mac)"
```

```
    case else
```

```
        game_text.Text = "Game: Unknown"
```

```
    end select
```

```
    select case main.maps_expanded(map_index).header.maptype
```

```
    case 0
```

```
        maptype_text.Text = "Map Type: Campaign"
```

```
    case 1
```

```
        maptype_text.Text = "Map Type: Multiplayer"
```

```
    case 2
```

```
        maptype_text.Text = "Map Type: User Interface"
```

```
    case else
```

```
        maptype_text.Text = "Map Type: Unknown"
```

```
    end select
```

```
    name_text.text = "Name: " + main.maps_expanded(map_index).header.name.replaceAll(chr(0), "")
```

```
    build_text.text = "Build: " + main.maps_expanded(map_index).header.builddate.replaceAll(chr(0), "")
```

```
    tagcount_text.text = "Tag Count: " + str(UBound(main.maps_expanded(map_index).tags) + 1)
```

```
    change_ok = false
```

```
    PopupMenu1.DeleteAllRows
```

```
    redim row_tags(-1)
```

```
    for i as integer = 0 to UBound(main.extracted_meta_pack)
```

```
        if main.extracted_meta_pack(i).metas.ubound > 0 then
```

```
            PopupMenu1.AddRow(main.extracted_meta_pack(i).class1.reverse + ": " + main.extracted_meta_pack(i).name  
+ " (Recursive)")
```

```
            row_tags.append i
```

```
        else
```

```
            PopupMenu1.AddRow(main.extracted_meta_pack(i).class1.reverse + ": " +  
main.extracted_meta_pack(i).name)
```

```
            row_tags.append i
```

```
        end
```

```
    next
```

```
    if PopupMenu1.ListCount > 0 then
```

```
        PopupMenu1.ListIndex = 0
```

```
    end
```

```
change_ok = true
End Sub
change_ok As boolean
```

```
main As main_Window
```

```
map_index As Integer
```

```
row_tags() As Integer
```

### **map\_rebuild\_control\_old\_old Control BevelButton1:**

```
Sub Action()
    Dim file As FolderItem
    file=GetSaveFolderItem(H1_Filetypes.HaloMapFile,"build.map")
    If file<> Nil then
        dim writer as new H1.map_writer
        dim temp_map as h1.map = main.maps_expanded(Map_Index)
        writer.init(temp_map, file, main)
        writer.run
        main.maps_expanded(Map_Index) = temp_map
    end if
End Sub
```

### **map\_rebuild\_control\_old\_old Control PushButton1:**

```
Sub Action()
    if change_ok and PopupMenu1.ListIndex >= 0 then
        dim row as integer = PopupMenu1.ListIndex
        dim index as integer = row_tags(row)
        'dim temp_pack as H1.Meta_Pack = main.extracted_meta_pack(index)
        'dim temp as integer
        dim temp_bool as boolean =
        main.maps_expanded(Map_Index).addtags(main.extracted_meta_pack(index).metas)
        if temp_bool then
            MsgBox("Tags Imported Successfully!")
        else
            errorbox("Error: Could not import tags")
        end
        update
    end
End Sub
End Class
```

## **Class map\_import\_thread**

Inherits Thread

### **map\_import\_thread.Run:**

```
Sub Run()
    dim temp_bool as boolean = true
    for i as integer = 0 to UBound(breaks)
        if temp_bool then
            dim temp_metas(-1) as h1.meta
```

```

    if i < breaks.Ubound then
        for j as integer = breaks(i) to breaks(i+1) - 1
            dim temp_meta as new h1.meta
            temp_meta = metas(j).copy
            temp_metas.append temp_meta
        next
    else
        for j as integer = breaks(i) to metas.Ubound
            dim temp_meta as new h1.meta
            temp_meta = metas(j).copy
            temp_metas.append temp_meta
        next
    end
    temp_bool = main.maps_expanded(Map_Index).addtags(temp_metas, no_duplicates)
end
next
if temp_bool then
    MsgBox("Tags Imported Successfully!")
else
    errorbox("Error: Could not import tags")
end
rebuild_control.update
main.update_list
End Sub

```

### **map\_import\_thread.init:**

```

Sub init(byref in_main as main_window, byref in_rebuild_control as map_rebuild_control, byref in_metas() as h1.meta,
in_no_duplicates as boolean, in_map_index as integer, byref in_breaks() as integer)

```

```

    main = in_main
    rebuild_control = in_rebuild_control
    metas = in_metas
    no_duplicates = in_no_duplicates
    Map_Index = in_map_index
    breaks = in_breaks
End Sub

```

```

breaks() As Integer

```

```

main As Main_Window

```

```

map_index As Integer

```

```

metas() As h1.meta

```

```

no_duplicates As boolean

```

```

rebuild_control As map_rebuild_control

```

```

End Class

```

## **Class map\_internalize\_thread**

```

Inherits Thread

```

map\_internalize\_thread.Run:

Sub Run()

```
dim status_denom as double = max(UBound(bitmap_inds),0) + max(UBound(sound_inds),0) + 1
dim missed_tags(-1) as integer
dim w as new Progress_Window("Internalizing Data")
w.tick("Internalizing bitmaps", 0)
for i as integer = 0 to UBound(bitmap_inds)
    dim status_num as double = i+1
    dim status_dec as double = status_num / status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Internalizing bitmaps", status)
    dim temp_bool as boolean = main.maps_expanded(map_index).internalizeTag(bitmap_inds(i), bitmaps_f)
    if not temp_bool then
        missed_tags.append bitmap_inds(i)
    end
next
for i as integer = 0 to UBound(sound_inds)
    dim status_num as double = i+1+UBound(bitmap_inds)
    dim status_dec as double = status_num / status_denom
    dim status as integer = status_dec * 100.0
    w.tick("Internalizing sounds", status)
    dim temp_bool as boolean = main.maps_expanded(map_index).internalizeTag(sound_inds(i), sounds_f)
    if not temp_bool then
        missed_tags.append sound_inds(i)
    end
next
w.close
if missed_tags.Ubound < 0 then
    MsgBox("Data Internalized Successfully")
else
    dim temp_str as string = ""
    for i as integer = 0 to UBound(missed_tags)
        temp_str = temp_str + EndOfLine + main.maps_expanded(map_index).tags(missed_tags(i)).class1.reverse +
        ":" + _
        main.maps_expanded(map_index).tags(missed_tags(i)).name
    next
    errorbox("All but the following tags were internalized:" + temp_str)
end
rebuilder.Update
End Sub
```

### map\_internalize\_thread.init:

```
Sub init(byref in_rebuilder as map_rebuild_control, byref in_main as Main_Window, in_map_index as integer,
in_bitmaps_f as folderItem, in_sounds_f as folderItem, in_bitmap_inds() as integer, in_sound_inds() as integer)
    rebuilder = in_rebuilder
    main = in_main
    map_index = in_map_index
    bitmaps_f = in_bitmaps_f
    sounds_f = in_sounds_f
    bitmap_inds = in_bitmap_inds
    sound_inds = in_sound_inds
End Sub
bitmaps_f As folderitem
```

bitmap\_inds() As Integer

main As Main\_Window

map\_index As integer

rebuilder As map\_rebuild\_control

sounds\_f As folderitem

sound\_inds() As Integer

End Class

## **Class map BSP replace thread**

Inherits Thread

### **map\_BSP\_replace\_thread.Run:**

Sub Run()

dim temp\_bool as boolean = main.maps\_expanded(map\_index).replaceSBSP(replacement\_index, \_  
main.extracted\_meta\_pack(meta\_pack\_index), no\_duplicates)

if temp\_bool then

MsgBox("BSP Succesfully Replaced!")

else

errorbox("Unable to replace BSP")

end

rebuilder.Update

End Sub

### **map\_BSP\_replace\_thread.init:**

Sub init(byref in\_main as main\_window, byref in\_rebuild\_control as map\_rebuild\_control, in\_map\_index as integer,  
in\_pack\_index as integer, in\_replacement\_index as integer, in\_no\_duplicates as boolean)

main = in\_main

rebuilder = in\_rebuild\_control

map\_index = in\_map\_index

meta\_pack\_index = in\_pack\_index

replacement\_index = in\_replacement\_index

no\_duplicates = in\_no\_duplicates

End Sub

main As main\_Window

map\_index As Integer

meta\_pack\_index As Integer

no\_duplicates As boolean

rebuilder As map\_rebuild\_control

replacement\_index As Integer



End Class

## **Class map\_rebuild\_full\_control**

Inherits ContainerControl

### **map\_rebuild\_full\_control.constructor:**

Sub constructor(byref in\_w as main\_window, in\_map\_index as integer)

    main = in\_w

    map\_index = in\_map\_index

    update

End Sub

### **map\_rebuild\_full\_control.update:**

Sub update()

    //find each of the tags and autocomplete the base tag form, then update the sbasp form

    change\_ok = false

    //scnr, find the base tag for the map

    dim temp\_id as uint32 = main.maps\_expanded(map\_index).index\_header.basetag

    if main.maps\_expanded(map\_index).tagIDTable.haskey(temp\_id) then

        //scnr tag is properly in index

        scnr\_ind = main.maps\_expanded(Map\_Index).tagIDTable.value(temp\_id)

        if main.maps\_expanded(map\_index).tags(scnr\_ind).class1 <> "rnccs" then scnr\_ind = -1

    else

        //scnr tag is not listed as the base tag

        scnr\_ind = -1

    end

    if scnr\_ind < 0 then

        //display nil info on scenario tag

        text\_scnr.text = "Select a tag to act as the scnr tag"

    else

        //display the scenario tag data

        text\_scnr.text = "scnr: " + main.maps\_expanded(map\_index).tags(scnr\_ind).name

    end

    //matg: globals\globals

    dim class\_str as string = "matg"

    dim name\_str as string = "globals\globals"

    if main.maps\_expanded(map\_index).tagClassNameTable.haskey(class\_str.reverse+":"+name\_str) then

        matg\_ind = main.maps\_expanded(Map\_Index).tagClassNameTable.value(class\_str.reverse+":"+name\_str)

    else

        matg\_ind = -1

    end

    if matg\_ind < 0 then

        text\_matg.text = "Select a tag to act as: " + class\_str+": "+name\_str

    else

        //display the scenario tag data

        text\_matg.text = class\_str+": "+name\_str

    end

```

//tagc: ui\ui_tags_loaded_all_scenario_types
class_str = "tagc"
name_str = "ui\ui_tags_loaded_all_scenario_types"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+":"+name_str) then
    tagc_all_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+":"+name_str)
else
    tagc_all_ind = -1
end
if tagc_all_ind < 0 then
    text_tagc_all.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_tagc_all.text = class_str+": "+name_str
end

//bitm: ui\shell\bitmaps\background
class_str = "bitm"
name_str = "ui\shell\bitmaps\background"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+":"+name_str) then
    bitm_background_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse
+":"+name_str)
else
    bitm_background_ind = -1
end
if bitm_background_ind < 0 then
    text_bitm_background.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_bitm_background.text = class_str+": "+name_str
end

//ustr: ui\shell\strings\loading
class_str = "ustr"
name_str = "ui\shell\strings\loading"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+":"+name_str) then
    ustr_loading_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+":"+name_str)
else
    ustr_loading_ind = -1
end
if ustr_loading_ind < 0 then
    text_ustr_loading.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_ustr_loading.text = class_str+": "+name_str
end

//ustr: ui\shell\main_menu\mp_map_list
class_str = "ustr"
name_str = "ui\shell\main_menu\mp_map_list"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+":"+name_str) then
    ustr_mp_map_list_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse
+":"+name_str)
else
    ustr_mp_map_list_ind = -1
end

```

```

end
if ustr_mp_map_list_ind < 0 then
    text_ustr_mp_map_list.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_ustr_mp_map_list.text = class_str+": "+name_str
end

//bitm: ui\shell\bitmaps\trouble_brewing
class_str = "bitm"
name_str = "ui\shell\bitmaps\trouble_brewing"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+": "+name_str) then
    bitm_trouble_brewing_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+": "+name_str)
else
    bitm_trouble_brewing_ind = -1
end
if bitm_trouble_brewing_ind < 0 then
    text_bitm_trouble_brewing.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_bitm_trouble_brewing.text = class_str+": "+name_str
end

//snd!: sound\sfx\ui\cursor
class_str = "snd!"
name_str = "sound\sfx\ui\cursor"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+": "+name_str) then
    snd_cursor_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+": "+name_str)
else
    snd_cursor_ind = -1
end
if snd_cursor_ind < 0 then
    text_snd_cursor.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_snd_cursor.text = class_str+": "+name_str
end

//snd!: sound\sfx\ui\forward
class_str = "snd!"
name_str = "sound\sfx\ui\forward"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+": "+name_str) then
    snd_forward_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+": "+name_str)
else
    snd_forward_ind = -1
end
if snd_forward_ind < 0 then
    text_snd_forward.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_snd_forward.text = class_str+": "+name_str
end

```

```

//snd!: sound\sfx\ui\back
class_str = "snd!"
name_str = "sound\sfx\ui\back"
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+": "+name_str) then
    snd_back_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+": "+name_str)
else
    snd_back_ind = -1
end
if snd_forward_ind < 0 then
    text_snd_back.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_snd_back.text = class_str+": "+name_str
end

//tagc: ui\ui_tags_loaded_multiplayer_scenario_type | ui\ui_tags_loaded_solo_scenario_type
// or ui\ui_tags_loaded_mainmenu_scenario_type
class_str = "tagc"
select case main.maps_expanded(Map_Index).header.maptype
case 0 //SP
    name_str = "ui\ui_tags_loaded_solo_scenario_type"
case 1 //MP
    name_str = "ui\ui_tags_loaded_multiplayer_scenario_type"
case 2 //UI
    name_str = "ui\ui_tags_loaded_mainmenu_scenario_type"
case else
    name_str = "unknown scenario type"
end select
if main.maps_expanded(map_index).tagClassNameTable.haskey(class_str.reverse+": "+name_str) then
    tagc_type_ind = main.maps_expanded(Map_Index).tagClassNameTable.value(class_str.reverse+": "+name_str)
else
    tagc_type_ind = -1
end
if tagc_type_ind < 0 then
    text_tagc_type.text = "Select a tag to act as: " + class_str+": "+name_str
else
    //display the scenario tag data
    text_tagc_type.text = class_str+": "+name_str
end

//sbsp, deal with it separately
if scnr_ind < 0 then
    //no scnr means no bsp data
    bsp_popup.DeleteAllRows
    bsp_list.DeleteAllRows
    bsp_dep_check.Enabled = false
    bsp_text.Enabled = false
    bsp_popup.Enabled = false
    bsp_list.Enabled = false
    bsp_push.Enabled = false
else
    bsp_popup.DeleteAllRows
    bsp_list.DeleteAllRows
    bsp_dep_check.Enabled = true

```

```

bsp_text.Enabled = true
bsp_popup.Enabled = true
bsp_list.Enabled = true
bsp_push.Enabled = true

//trust the map bsp_data
//make sure that if the scnr data is updated to reupdate the map bsp_data
for i as integer = 0 to main.maps_expanded(map_index).bsp_data.ubound
    if main.maps_expanded(map_index).tagIDTable.HasKey(main.maps_expanded(map_index).bsp_data(i).tagID)
    then
        dim bsp_ind as integer = _
        main.maps_expanded(map_index).tagIDTable.value(main.maps_expanded(map_index).bsp_data(i).tagID)
        bsp_popup.AddRow("BSP " + str(i+1) + ": SBSP: " + main.maps_expanded(map_index).tags(bsp_ind).name)
    else
        break
    end
next
if main.maps_expanded(map_index).bsp_data.ubound >= 0 then
    bsp_popup.ListIndex = 0
end

dim str_list(-1) as string
dim ind_list(-1) as integer
for i as integer = 0 to main.extracted_meta_pack.Ubound
    dim has_bsp as boolean = false
    for j as integer = 0 to main.extracted_meta_pack(i).metas.ubound
        if main.extracted_meta_pack(i).metas(j).class1 = "psbs" then
            has_bsp = true
            exit for j
        end
    next
    if has_bsp then
        if main.extracted_meta_pack(i).metas.ubound = 0 then
            str_list.Append main.extracted_meta_pack(i).class1.reverse + main.extracted_meta_pack(i).name
        else
            str_list.Append main.extracted_meta_pack(i).class1.reverse + main.extracted_meta_pack(i).name +
            " (Recursive)"
        end
        ind_list.Append i
    end
next

str_list.SortWith(ind_list)

for i as integer = 0 to UBound(ind_list)
    if main.extracted_meta_pack(ind_list(i)).metas.ubound = 0 then
        bsp_list.AddRow(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
        main.extracted_meta_pack(ind_list(i)).name)
        bsp_list.CellTag(bsp_list.LastIndex, 0) = "t:" + str(ind_list(i)) + "_" + str(0)
    else
        bsp_list.AddFolder(main.extracted_meta_pack(ind_list(i)).class1.reverse + ": " +
        main.extracted_meta_pack(ind_list(i)).name + " (Recursive)")
        bsp_list.CellTag(bsp_list.LastIndex, 0) = "p:" + str(ind_list(i))
    end
end

```

```

    next

end

    change_ok = true
End Sub
bitm_background_ind As Integer

bitm_trouble_brewing_ind As Integer

change_ok As boolean

main As main_window

map_index As Integer

matg_ind As Integer

scnr_ind As Integer

snd_back_ind As Integer

snd_cursor_ind As Integer

snd_forward_ind As Integer

tagc_all_ind As Integer

tagc_type_ind As Integer

ustr_loading_ind As Integer

ustr_mp_map_list_ind As Integer

```

### **map\_rebuild\_full\_control Note: tags**

```

tags
scnr
matg: globals\globals
tagc: ui\ui_tags_loaded_all_scenario_types
bitm: ui\shell\bitmaps\background
ustr: ui\shell\strings\loading
ustr: ui\shell\main_menu\mp_map_list
bitm: ui\shell\bitmaps\trouble_brewing
snd!: sound\sfx\ui\cursor
snd!: sound\sfx\ui\forward
snd!: sound\sfx\ui\back
tagc: ui\ui_tags_loaded_multiplayer_scenario_type | ui\ui_tags_loaded_solo_scenario_type
sbsp

```

### **map\_rebuild\_full\_control Note: what to do**

```

what to do
//Base Tags
option to alter what the base level tags are

```

//BSP Selection

how to change which BSP the current scnr file uses

//Tag Removal

the option to remove a tag or set of tags and null all dependencies

garbage collect, remove any tags not part referenced by any of the base tags

instead of tag removal:

during rebuild the option to not include orphaned tags

**map\_rebuild\_full\_control Control bsp\_list:**

Sub ExpandRow(row As Integer)

if not change\_ok then return

dim temp\_split() as string = me.CellTag(row, 0).split(":")

if temp\_split.Ubound < 1 then return

select case temp\_split(0)

case "m"

//should never fire

case "p"

//meta pack, expand it

dim index as integer = val(temp\_split(1))

dim inds(-1) as integer

dim strings(-1) as string

for i as integer = 0 to UBound(main.extracted\_meta\_pack(index).metas)

if main.extracted\_meta\_pack(index).metas(i).class1 = "psbs" then //only include sbsp tags

inds.append i

strings.append main.extracted\_meta\_pack(index).metas(i).class1.reverse + ": " +

main.extracted\_meta\_pack(index).metas(i).name

end

next

strings.SortWith(inds)

for i as integer = 0 to UBound(inds)

me.AddRow(main.extracted\_meta\_pack(index).metas(inds(i)).class1.reverse + ": " +

main.extracted\_meta\_pack(index).metas(inds(i)).name)

me.CellTag(me.LastIndex, 0) = "t:" + str(index) + "\_" + str(inds(i))

next

case else

//should never fire

end select

End Sub

End Class

## **Class extracted\_tag\_pack\_control**

Inherits ContainerControl

### **extracted\_tag\_pack\_control.Constructor:**

Sub Constructor(in\_w as main\_Window, in\_extracted\_meta\_pack\_index as integer)

main = in\_w

extracted\_meta\_pack\_index = in\_extracted\_meta\_pack\_index

update

End Sub

### **extracted\_tag\_pack\_control.update:**

Sub update()

```
if main.extracted_meta_pack(extracted_meta_pack_index).metas.ubound > -1 then
    tag_text.Text = main.extracted_meta_pack(extracted_meta_pack_index).metas(0).class1.reverse + ": " _
    + main.extracted_meta_pack(extracted_meta_pack_index).metas(0).name
end
if main.extracted_meta_pack(extracted_meta_pack_index).metas.ubound > 0 then
    //dependent tags
    for i as integer = 1 to main.extracted_meta_pack(extracted_meta_pack_index).metas.ubound
        Listbox1.AddRow( main.extracted_meta_pack(extracted_meta_pack_index).metas(i).class1.reverse)
        Listbox1.Cell(Listbox1.LastIndex, 1) = main.extracted_meta_pack(extracted_meta_pack_index).metas(i).name
    next
else
    GroupBox2.Enabled = false
end
```

self.Refresh

End Sub

extracted\_meta\_pack\_index As Integer

main As main\_Window

End Class

## **Class expanded\_tag\_control**

Inherits ContainerControl

### **expanded\_tag\_control.change\_name:**

Sub change\_name(new\_name as string)

```
main.maps_expanded(map_index).change_name(tag_index, new_name)
if not main.pref.byName then
    main.update_list
else
    //do something to handle by name issues
    for i as integer = 0 to main.ListBox1.ListCount-1
        if main.ListBox1.Expanded(i) then
            main.ListBox1.Expanded(i) = false
        end
    next
end
update
End Sub
```

### **expanded\_tag\_control.Constructor:**

Sub Constructor(in\_w as main\_window, in\_map\_index as integer, in\_tag\_index as integer)

```
main = in_w
map_index = in_map_index
tag_index = in_tag_index
```

End Sub



**expanded\_tag\_control.find\_referencing\_tags:**

Function find\_referencing\_tags() As Integer()

dim this\_tag\_class as string = main.maps\_expanded(map\_index).tags(tag\_index).class1

dim this\_tag\_name as string = main.maps\_expanded(map\_index).tags(tag\_index).name

dim ref\_tag\_inds(-1) as integer

for i as integer = 0 to main.maps\_expanded(map\_index).tags.ubound

if main.maps\_expanded(map\_index).tags(i).expanded then

for j as integer = 0 to main.maps\_expanded(map\_index).tags(i).data.dependencies.ubound

if main.maps\_expanded(map\_index).tags(i).data.dependencies(j).tag\_class = this\_tag\_class and \_

main.maps\_expanded(map\_index).tags(i).data.dependencies(j).tag\_name = this\_tag\_name then

ref\_tag\_inds.Append i

continue for i

end

next

for j as integer = 0 to main.maps\_expanded(map\_index).tags(i).data.loneIDs.ubound

if main.maps\_expanded(map\_index).tags(i).data.loneIDs(j).tag\_class = this\_tag\_class and \_

main.maps\_expanded(map\_index).tags(i).data.loneIDs(j).tag\_name = this\_tag\_name then

ref\_tag\_inds.Append i

continue for i

end

next

end

next

ref\_tag\_inds = remove\_redundant\_ints(ref\_tag\_inds)

return ref\_tag\_inds

End Function

**expanded\_tag\_control.update:**

Sub update()

dim name\_str as string = main.maps\_expanded(map\_index).tags(tag\_index).name

name\_edit.Text = name\_str

dim referencing\_tags() as integer = find\_referencing\_tags

ref\_list.DeleteAllRows

for i as integer = 0 to referencing\_tags.Ubound

dim class\_str as string = main.maps\_expanded(map\_index).tags(referencing\_tags(i)).class1.reverse

dim ref\_name\_str as string = main.maps\_expanded(map\_index).tags(referencing\_tags(i)).name

ref\_list.AddRow(class\_str)

ref\_list.Cell(ref\_list.LastIndex, 1) = ref\_name\_str

ref\_list.CellTag(ref\_list.LastIndex, 0) = referencing\_tags(i)

next

self.Refresh

End Sub

last\_row As Integer

main As Main\_Window

map\_index As Integer

tag\_index As Integer

**expanded\_tag\_control Control name\_edit:**

Function KeyDown(Key As String) As Boolean

```

    if asc(key) = 3 or asc(key) = 13 then
        dim new_name as string = name_edit.text
        change_name(new_name)
    end
End Function

```

### **expanded\_tag\_control Control name\_change\_push:**

```

Sub Action()
    dim new_name as string = name_edit.text
    change_name(new_name)
End Sub

```

### **expanded\_tag\_control Control ref\_list:**

```

Sub DoubleClick()
    if last_row = -1 then return
    dim referenced_index as integer = ref_list.CellTag(last_row, 0)
    dim class_str as string = main.maps_expanded(map_index).tags(referenced_index).class1.reverse
    dim name_str as string = main.maps_expanded(map_index).tags(referenced_index).name
    dim temp_bool as boolean = main.select_tag("maps_expanded:" + str(map_index), class_str, name_str)
End Sub

```

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    last_row = row
End Function

```

### **expanded\_tag\_control Control BevelButton1:**

```

Sub Action()
    dim w as new Tag_Delete_Window(main, map_index, tag_index)
    w.show
End Sub
End Class

```

## **Class dependency\_swapper\_control\_RW**

Inherits ContainerControl

### **dependency\_swapper\_control\_RW.Constructor:**

```

Sub Constructor(in_f as folderItem, in_map as h1.map, in_offset as integer, in_size as integer, in_parent as
main_Window, in_map_index as integer, in_tagID as uint32)
    f = in_f
    map = in_map
    offset = in_offset
    size = in_size
    change_ok = false
    parent_window = in_parent
    map_index = in_map_index
    this_tagID = in_tagID
End Sub

```

### **dependency\_swapper\_control\_RW.read:**

```

Sub read()
    redim dep_offset(-1)
    redim dep_stroffset(-1)

```

```

redim dep_tagid(-1)
redim dep_class(-1)
redim dep_name(-1)

redim lone_offset(-1)
redim lone_tagid(-1)
redim lone_class(-1)
redim lone_name(-1)

dim br as BinaryStream = f.OpenAsBinaryFile
br.LittleEndian = true
br.Position = offset

'dim start_tick as integer = ticks

while br.position < offset + size
    dim temp_position as integer = br.position
    dim temp_val as UInt32 = br.ReadUInt32
    if map.tagIDTable.haskey(temp_val) then
        //either a loneID or dependency
        br.Position = temp_position - 4 - 4 -4
        dim class_str as string = reverse(br.read(4))
        if map.cTagtypes.haskey(class_str) then
            //dependency
            dep_class.append reverse(map.tags(map.tagIDTable.value(temp_val)).class1)
            dep_offset.append temp_position -4 -4 -4
            dep_stroffset.Append map.tags(map.tagIDTable.value(temp_val)).nameoffset
            dep_name.Append map.tags(map.tagIDTable.value(temp_val)).name
            dep_tagid.Append temp_val
        else
            //loneID
            lone_offset.append temp_position
            lone_tagid.append temp_val
            lone_class.append reverse(map.tags(map.tagIDTable.value(temp_val)).class1)
            lone_name.append map.tags(map.tagIDTable.value(temp_val)).name
        end
        br.position = temp_position + 4
    end
    if temp_val = &hFFFFFFF then
        //might be a dependency
        br.Position = temp_position - 4 - 4 -4
        dim class_str as string = reverse(br.read(4))
        if map.cTagtypes.haskey(class_str) then
            //dependency
            dep_class.append class_str
            dep_offset.append temp_position -4 -4 -4
            dep_stroffset.Append br.ReadInt32
            dep_name.Append "nulled out"
            dep_tagid.Append temp_val
        end
        br.Position = temp_position + 4
    end
wend
br.Close

```

```
'dim time as integer = ticks - start_tick
'dim break_point as integer
```

```
update
End Sub
```

### **dependency\_swapper\_control\_RW.swap:**

```
Sub swap(do_update as boolean = true)
    dim tag_info() as string = split(selected_tag, ":")
    if UBound(tag_info) < 1 then Return
    dim tagID as uint32 = PopupMenu1.RowTag(PopupMenu1.ListIndex)
    if tag_info(0) = "d" then
        dim magic as integer = map.magic
        if map.tagIDTable.haskey(this_tagID) then
            if map.tags(map.tagIDTable.value(this_tagID)).class1 = "psbs" then
                for i as integer = 0 to UBound(map.bsp_data)
                    if map.bsp_data(i).tagID = this_tagID then
                        magic = map.bsp_data(i).magic
                        exit for i
                    end
                next
            end
        end
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = dep_offset(val(tag_info(1)))
        if map.tagIDTable.haskey(tagID) then
            bw.write(map.tags(map.tagIDTable.value(tagID)).class1)
            dep_class(val(tag_info(1))) = reverse(map.tags(map.tagIDTable.value(tagID)).class1)
            bw.writeint32(map.tags(map.tagIDTable.value(tagID)).nameoffset + magic)
            dep_stroffset(val(tag_info(1))) = map.tags(map.tagIDTable.value(tagID)).nameoffset
            dep_name(val(tag_info(1))) = map.tags(map.tagIDTable.value(tagID)).name
            bw.position = dep_offset(val(tag_info(1))) + 4 + 4 + 4
        else
            bw.position = dep_offset(val(tag_info(1))) + 4 + 4 + 4
            dep_name(val(tag_info(1))) = "nulled out"
        end
        bw.WriteUInt32(tagID)
        bw.Close
        dep_tagid(val(tag_info(1))) = tagID
        if do_update then
            update
        end
    end
    if tag_info(0) = "l" then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = lone_offset(val(tag_info(1)))
        bw.WriteUInt32(tagID)
        bw.Close
        if map.tagIDTable.haskey(tagID) then
            lone_class(val(tag_info(1))) = reverse(map.tags(map.tagIDTable.value(tagID)).class1)
            lone_name(val(tag_info(1))) = map.tags(map.tagIDTable.value(tagID)).name
        end
    end
end
```

```

    else
        lone_name(val(tag_info(1))) = "nulled out"
    end
    lone_tagid(val(tag_info(1))) = tagID
    if do_update then
        update
    end
end
End Sub

```

### **dependency\_swapper\_control\_RW.tag\_selected:**

```

Sub tag_selected()
    if selected_tag = "" then Return

    dim tag_info() as string = split(selected_tag, ":")
    if UBound(tag_info) < 1 then Return

    update_tag_class_list

    if tag_info(0) = "d" then
        change_ok = false
        dim tag_class_str as string = dep_class(val(tag_info(1)))
        for i as integer = 0 to tag_class_popup.ListCount - 1
            if tag_class_popup.RowTag(i) = tag_class_str then
                tag_class_popup.ListIndex = i
            end
        next
        update_string_list(dep_class(val(tag_info(1))), dep_tagid(val(tag_info(1))))
        change_ok = true
    end

    if tag_info(0) = "l" then
        change_ok = false
        dim tag_class_str as string = lone_class(val(tag_info(1)))
        for i as integer = 0 to tag_class_popup.ListCount - 1
            if tag_class_popup.RowTag(i) = tag_class_str then
                tag_class_popup.ListIndex = i
            end
        next
        update_string_list(lone_class(val(tag_info(1))), lone_tagid(val(tag_info(1))))
        change_ok = true
    end
End Sub

```

### **dependency\_swapper\_control\_RW.update:**

```

Sub update()
    change_ok = false
    Listbox1.DeleteAllRows
    tag_class_popup.DeleteAllRows
    PopupMenu1.DeleteAllRows
    dim max_offset_chr_len as integer = 0
    for i as integer = 0 to dep_offset.ubound
        max_offset_chr_len = max(max_offset_chr_len, _
            hex(dep_offset(i)).Len)
    next

```

```

next
for i as integer = 0 to lone_offset.ubound
    max_offset_chr_len = max(max_offset_chr_len, _
    hex(lone_offset(i)).Len)
next
for i as integer = 0 to UBound(dep_tagid)
    Listbox1.AddRow(dep_class(i))
    Listbox1.Cell(Listbox1.LastIndex, 1) = dep_name(i)
    dim offset_str as string = hex(dep_offset(i))
    while offset_str.len < max_offset_chr_len
        offset_str = "0" + offset_str
    wend
    Listbox1.Cell(Listbox1.LastIndex, 2) = offset_str
    Listbox1.CellTag(Listbox1.LastIndex, 0) = "d:" + str(i)
    Listbox1.CellTag(Listbox1.LastIndex, 1) = "d:" + str(i)
    Listbox1.CellTag(Listbox1.LastIndex, 2) = "d:" + str(i)
next
for i as integer = 0 to UBound(lone_tagid)
    Listbox1.AddRow(lone_class(i))
    Listbox1.Cell(Listbox1.LastIndex, 1) = lone_name(i)
    dim offset_str as string = hex(lone_offset(i))
    while offset_str.len < max_offset_chr_len
        offset_str = "0" + offset_str
    wend
    Listbox1.Cell(Listbox1.LastIndex, 2) = offset_str
    Listbox1.CellTag(Listbox1.LastIndex, 0) = "l:" + str(i)
    Listbox1.CellTag(Listbox1.LastIndex, 1) = "l:" + str(i)
    Listbox1.CellTag(Listbox1.LastIndex, 2) = "l:" + str(i)
next
'for i as integer = 0 to Listbox1.ListCount - 1
'if Listbox1.CellTag(i, 0) = selected_tag then
'Listbox1.ListIndex = i
'exit for i
'end
'next
//leave off the whole selected_tag selection crap for now
Listbox1.PressHeader(sort_column)
change_ok = true
End Sub

```

### **dependency\_swapper\_control\_RW.update\_string\_list:**

```

Sub update_string_list(class_str as string, tagID as uint32)
    change_ok = false
    PopupMenu1.DeleteAllRows
    PopupMenu1.AddRow("nulled out")
    PopupMenu1.RowTag(0) = -1
    PopupMenu1.ListIndex = 0
    dim class_index as integer = -1
    for i as integer = 0 to UBound(map.meta_class_folders.child)
        if map.meta_class_folders.child(i).name.mid(1,4).instr(class_str) > 0 then
            class_index = i
            exit for i
        end
    next

```

```

if class_index = -1 then
    change_ok = true
    return
end
dim tag_indexes(-1) as integer
dim tag_strs(-1) as string
for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
    dim index as integer = map.meta_class_folders.child(class_index).item(i)
    tag_indexes.append index
    tag_strs.append map.tags(index).name
next
tag_strs.SortWith(tag_indexes)
for i as integer = 0 to UBound(tag_indexes)
    PopupMenu1.AddRow(map.tags(tag_indexes(i)).name)
    PopupMenu1.RowTag(i+1) = map.tags(tag_indexes(i)).tagID
    if map.tags(tag_indexes(i)).tagID = tagID then
        PopupMenu1.ListIndex = i + 1
    end
next
change_ok = true
End Sub

```

### **dependency\_swapper\_control\_RW.update\_tag\_class\_list:**

```

Sub update_tag_class_list()
    change_ok = false

    tag_class_popup.DeleteAllRows
    dim tag_class_names(-1) as string
    dim temp_keys(-1) as Variant
    temp_keys = map.cTagTypes.keys
    for i as integer = 0 to UBound(temp_keys)
        if VarType(temp_keys(i)) = 8 then
            tag_class_names.Append temp_keys(i).StringValue
        end
    next
    tag_class_names.Sort
    for i as integer = 0 to UBound(tag_class_names)
        tag_class_popup.AddRow(tag_class_names(i))
        tag_class_popup.RowTag(i) = tag_class_names(i)
    next

    change_ok = true
End Sub
change_ok As boolean

dep_class() As string

dep_name() As string

dep_offset() As Integer

dep_stroffset() As Integer

dep_tagid() As uint32

```

f As folderitem

last\_column As Integer

last\_row As Integer

lone\_class() As string

lone\_name() As string

lone\_offset() As Integer

lone\_tagid() As uint32

map As H1.map

map\_index As Integer

offset As Integer

parent\_window As main\_Window

selected\_tag As string

size As Integer

sort\_column As Integer

this\_tagID As uint32

## **dependency\_swapper\_control\_RW Control ListBox1:**

Sub DoubleClick()

if not change\_ok then Return

dim index\_str as string = me.CellTag(last\_row, 1)

dim index\_strs() as string = split(index\_str, ":")

dim class\_str as string

dim name\_str as string

dim null\_flag as boolean = false

if index\_strs(0) = "d" then

class\_str = dep\_class(val(index\_strs(1)))

name\_str = dep\_name(val(index\_strs(1)))

if dep\_tagid(val(index\_strs(1))) = &hFFFFFFFF then

null\_flag = true

end

end

if index\_strs(0) = "l" then

class\_str = lone\_class(val(index\_strs(1)))

name\_str = lone\_name(val(index\_strs(1)))

if lone\_tagid(val(index\_strs(1))) = &hFFFFFFFF then

null\_flag = true

end



```

end
if not null_flag then
    dim result as boolean = parent_window.select_tag("maps_lite:" +str(map_index), class_str, name_str)
end
End Sub

```

```

Function Click(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    last_row = row
    last_column = column
    if change_ok then
        if listbox1.Selected(row) = false then
            selected_tag = me.CellTag(row, column)
            tag_selected
        end
    end
End Function

```

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31
        if listbox1.listindex + 1 < listbox1.ListCount then
            row = listbox1.ListIndex + 1
            ok = true
        end
    case 30
        if listbox1.listindex -1 > -1 then
            row = listbox1.ListIndex - 1
            ok = true
        end
    end
    if ok then
        last_row = row
        last_column = column
        if change_ok then
            if listbox1.Selected(row) = false then
                selected_tag = me.CellTag(row, column)
                tag_selected
            end
        end
    end
End Function

```

```

Function SortColumn(column As Integer) As Boolean
    sort_column = column
End Function

```

### **dependency\_swapper\_control\_RW Control PushButton1:**

```

Sub Action()
    if not change_ok then Return
    dim selected_tags() as string
    dim selected_indexes() as integer

```

```

for i as integer = 0 to ListBox1.ListCount - 1
    if Listbox1.Selected(i) then
        selected_tags.append listbox1.CellTag(i, 0)
        selected_indexs.append i
    end
next
for i as integer = 0 to UBound(selected_tags)
    selected_tag = selected_tags(i)
    swap(false)
next
update
for i as integer = 0 to UBound(selected_indexs)
    Listbox1.Selected(selected_indexs(i)) = true
next
if UBound(selected_indexs) > -1 then
    selected_tag = Listbox1.CellTag(selected_indexs(UBound(selected_indexs)), 0)
    tag_selected
end
End Sub

```

### **dependency\_swapper\_control\_RW Control tag\_class\_popup:**

```

Sub Change()
    if not change_ok then Return
    dim tag_info() as string = split(selected_tag, ":")
    if UBound(tag_info) < 1 then Return

    dim class_str as string = me.RowTag(me.ListIndex)

    if tag_info(0) = "d" then
        update_string_list(class_str, dep_tagid(val(tag_info(1))))
    end

    if tag_info(0) = "l" then
        update_string_list(class_str, lone_tagid(val(tag_info(1))))
    end
End Sub
End Class

```

## **Class dependency\_swapper\_control\_EX**

Inherits ContainerControl

### **dependency\_swapper\_control\_EX.Constructor:**

```

Sub Constructor(in_main as Main_Window, in_map_index as integer, in_tag_index as integer)
    change_ok = false
    main = in_main
    map_index = in_map_index
    tag_index = in_tag_index
End Sub

```

### **dependency\_swapper\_control\_EX.swap:**

```

Sub swap(do_update as boolean = true)
    dim tag_info() as string = split(selected_tag, ":")

```

```

if UBound(tag_info) < 1 then Return
dim ref_ind as integer = val(tag_info(1))
dim new_tag_info() as string = split(PopupMenu1.RowTag(PopupMenu1.ListIndex), ":")
if UBound(new_tag_info) < 1 then return
dim new_tag_class as string = new_tag_info(0)
dim new_tag_name as string = new_tag_info(1)
if tag_info(0) = "d" then
    if new_tag_class <> "LLUN" then
        main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_class = new_tag_class
    end
    main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_name= new_tag_name
    if do_update then
        update
    end
end
if tag_info(0) = "l" then
    main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_class = new_tag_class
    main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_name= new_tag_name
    if do_update then
        update
    end
end
End Sub

```

### **dependency\_swapper\_control\_EX.tag\_selected:**

```

Sub tag_selected()
    if selected_tag = "" then Return

    dim tag_info() as string = split(selected_tag, ":")
    if UBound(tag_info) < 1 then Return
    dim ref_ind as integer = val(tag_info(1))

    update_tag_class_list

    if tag_info(0) = "d" then
        change_ok = false
        dim tag_class_str as string =
            main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_class.reverse
        dim tag_name_str as string =
            main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_name
        for i as integer = 0 to tag_class_popup.ListCount - 1
            if tag_class_popup.RowTag(i) = tag_class_str then
                tag_class_popup.ListIndex = i
            end
        next
        update_string_list(tag_class_str, tag_name_str)
        change_ok = true
    end

    if tag_info(0) = "l" then
        change_ok = false
        dim tag_class_str as string =
            main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_class.reverse
        dim tag_name_str as string = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_name
    end

```

```

    for i as integer = 0 to tag_class_popup.ListCount - 1
        if tag_class_popup.RowTag(i) = tag_class_str then
            tag_class_popup.ListIndex = i
        end
    next
    update_string_list(tag_class_str, tag_name_str)
    change_ok = true
end
End Sub

```

### **dependency\_swapper\_control\_EX.update:**

```

Sub update()
    change_ok = false
    Listbox1.DeleteAllRows
    PopupMenu1.DeleteAllRows
    tag_class_popup.DeleteAllRows
    dim max_offset_chr_len as integer = 0
    for i as integer = 0 to main.maps_expanded(map_index).tags(tag_index).data.dependencies.ubound
        max_offset_chr_len = max(max_offset_chr_len, _
            hex(main.maps_expanded(map_index).tags(tag_index).data.dependencies(i).offset).Len)
    next
    for i as integer = 0 to main.maps_expanded(map_index).tags(tag_index).data.loneIDs.ubound
        max_offset_chr_len = max(max_offset_chr_len, _
            hex(main.maps_expanded(map_index).tags(tag_index).data.loneIDs(i).offset).Len)
    next
    for i as integer = 0 to main.maps_expanded(map_index).tags(tag_index).data.dependencies.ubound
        dim class_str as string =
            main.maps_expanded(map_index).tags(tag_index).data.dependencies(i).tag_class.reverse
        dim name_str as string = main.maps_expanded(map_index).tags(tag_index).data.dependencies(i).tag_name
        dim offset_str as string = hex(main.maps_expanded(map_index).tags(tag_index).data.dependencies(i).offset)
        while offset_str.len < max_offset_chr_len
            offset_str = "0" + offset_str
        wend
        Listbox1.AddRow(class_str)
        Listbox1.Cell(Listbox1.LastIndex, 1) = name_str
        Listbox1.Cell(Listbox1.LastIndex, 2) = offset_str
        Listbox1.CellTag(Listbox1.LastIndex, 0) = "d:" + str(i)
    next
    for i as integer = 0 to main.maps_expanded(map_index).tags(tag_index).data.loneIDs.ubound
        dim class_str as string = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(i).tag_class.reverse
        dim name_str as string = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(i).tag_name
        dim offset_str as string = hex(main.maps_expanded(map_index).tags(tag_index).data.loneIDs(i).offset)
        while offset_str.len < max_offset_chr_len
            offset_str = "0" + offset_str
        wend
        Listbox1.AddRow(class_str)
        Listbox1.Cell(Listbox1.LastIndex, 1) = name_str
        Listbox1.Cell(Listbox1.LastIndex, 2) = offset_str
        Listbox1.CellTag(Listbox1.LastIndex, 0) = "l:" + str(i)
    next
    Listbox1.PressHeader(sort_column)
    change_ok = true
End Sub

```

dependency\_swapper\_control\_EX.update\_string\_list:

```
Sub update_string_list(class_str as string, name_str as string)
    change_ok = false
    PopupMenu1.DeleteAllRows
    PopupMenu1.AddRow("nulled out")
    PopupMenu1.RowTag(0) = "LLUN:nulled out"
    PopupMenu1.ListIndex = 0
    dim class_index as integer = -1
    for i as integer = 0 to UBound(main.maps_expanded(map_index).meta_class_folders.child)
        if main.maps_expanded(map_index).meta_class_folders.child(i).name.mid(1,4).instr(class_str) > 0 then
            class_index = i
            exit for i
        end
    next
    if class_index = -1 then
        change_ok = true
        return
    end
    dim tag_indexs(-1) as integer
    dim tag_strs(-1) as string
    for i as integer = 0 to UBound(main.maps_expanded(map_index).meta_class_folders.child(class_index).item)
        dim index as integer = main.maps_expanded(map_index).meta_class_folders.child(class_index).item(i)
        tag_indexs.append index
        tag_strs.append main.maps_expanded(map_index).tags(index).name
    next
    tag_strs.SortWith(tag_indexs)
    for i as integer = 0 to UBound(tag_indexs)
        PopupMenu1.AddRow(main.maps_expanded(map_index).tags(tag_indexs(i)).name)
        dim row_tag_str as string = main.maps_expanded(map_index).tags(tag_indexs(i)).class1 _
        + ":" + main.maps_expanded(map_index).tags(tag_indexs(i)).name
        PopupMenu1.RowTag(i+1) = row_tag_str
        if main.maps_expanded(map_index).tags(tag_indexs(i)).class1.reverse = class_str _
            and main.maps_expanded(map_index).tags(tag_indexs(i)).name = name_str then
            PopupMenu1.ListIndex = i + 1
        end
    next
    change_ok = true
End Sub
```

**dependency\_swapper\_control\_EX.update\_tag\_class\_list:**

```
Sub update_tag_class_list()
    change_ok = false

    tag_class_popup.DeleteAllRows
    dim tag_class_names(-1) as string
    dim temp_keys(-1) as Variant
    temp_keys = main.maps_expanded(map_index).cTagTypes.keys
    for i as integer = 0 to UBound(temp_keys)
        if VarType(temp_keys(i)) = 8 then
            tag_class_names.Append temp_keys(i).StringValue
        end
    next
    tag_class_names.Sort
    for i as integer = 0 to UBound(tag_class_names)
```

```

        tag_class_popup.AddRow(tag_class_names(i))
        tag_class_popup.RowTag(i) = tag_class_names(i)
    next

    change_ok = true
End Sub
change_ok As boolean

last_column As Integer

last_row As Integer

```

### **dependency\_swapper\_control\_EX.main:**

```

main As Main_Window
//keep this

```

### **dependency\_swapper\_control\_EX.map\_index:**

```

map_index As Integer
//keep this
selected_tag As string

```

```

sort_column As Integer

```

### **dependency\_swapper\_control\_EX.tag\_index:**

```

tag_index As Integer
//keep this

```

### **dependency\_swapper\_control\_EX Control ListBox1:**

```

Function Click(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean

```

```

    last_row = row
    last_column = column
    if change_ok then
        if listbox1.Selected(row) = false then
            selected_tag = me.CellTag(row, 0)
            tag_selected
        end
    end
End Function

```

```

Function KeyDown(Key As String) As Boolean

```

```

    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31
        if listbox1.listindex + 1 < listbox1.ListCount then
            row = listbox1.ListIndex + 1
            ok = true
        end
    case 30
        if listbox1.listindex - 1 > -1 then

```

```

        row = listbox1.ListIndex - 1
        ok = true
    end
end
if ok then
    last_row = row
    last_column = column
    if change_ok then
        if listbox1.Selected(row) = false then
            selected_tag = me.CellTag(row, 0)
            tag_selected
        end
    end
end
End Function

```

```

Function SortColumn(column As Integer) As Boolean
    sort_column = column
End Function

```

```

Sub DoubleClick()
    if not change_ok then Return

    dim index_str as string = me.CellTag(last_row, 0)
    dim index_strs() as string = split(index_str, ":")
    dim ref_ind as integer = val(index_strs(1))
    dim class_str as string
    dim name_str as string
    dim null_flag as boolean = false
    if index_strs(0) = "d" then
        class_str = main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_class.reverse
        name_str = main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_name
        if name_str = "nulled out" then
            null_flag = true
        end
    end
    if index_strs(0) = "l" then
        class_str = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_class.reverse
        name_str = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_name
        if name_str = "nulled out" then
            null_flag = true
        end
    end
    if not null_flag then
        dim result as boolean = main.select_tag("maps_expanded:" + str(map_index), class_str, name_str)
    end
End Sub

```

### **dependency\_swapper\_control\_EX Control PushButton1:**

```

Sub Action()
    if not change_ok then Return
    dim selected_tags() as string
    dim selected_indexs() as integer
    for i as integer = 0 to ListBox1.ListCount - 1

```

```

        if Listbox1.Selected(i) then
            selected_tags.append listbox1.CellTag(i, 0)
            selected_indexs.append i
        end
    next
    for i as integer = 0 to UBound(selected_tags)
        selected_tag = selected_tags(i)
        swap(false)
    next
    update
    for i as integer = 0 to UBound(selected_indexs)
        Listbox1.Selected(selected_indexs(i)) = true
    next
    if UBound(selected_indexs) > -1 then
        selected_tag = Listbox1.CellTag(selected_indexs(UBound(selected_indexs)), 0)
        tag_selected
    end
End Sub

```

### **dependency\_swapper\_control\_EX Control tag\_class\_popup:**

```

Sub Change()
    if not change_ok then return

    dim tag_info() as string = split(selected_tag, ":")
    if UBound(tag_info) < 1 then Return
    dim ref_ind as integer = val(tag_info(1))

    if tag_info(0) = "d" then
        dim tag_name as string =
            main.maps_expanded(map_index).tags(tag_index).data.dependencies(ref_ind).tag_name
        update_string_list(me.text, tag_name)
    end

    if tag_info(0) = "l" then
        dim tag_name as string = main.maps_expanded(map_index).tags(tag_index).data.loneIDs(ref_ind).tag_name
        update_string_list(me.text, tag_name)
    end
End Sub
End Class

```

## **Class bitmask8\_editor\_control\_RWS**

Inherits ContainerControl

### **bitmask8\_editor\_control\_RWS.LostFocus:**

```

Sub LostFocus()
    me.read
End Sub

```

### **bitmask8\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset

```



```

    reflexive_offset = 0
    labels = in_labels
    values = in_values
End Sub

```

### **bitmask8\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt8
    br.Close

    Listbox1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        Listbox1.AddRow(labels(i))
        Listbox1.CellTag(i,0) = i
    next
    Listbox1.ColumnType(0) = 2

    dim temp_str as string = bin(value)
    while temp_str.len < 8
        temp_str = "0" + temp_str
    wend
    for i as integer = 0 to Listbox1.ListCount - 1
        if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
            Listbox1.CellCheck(i,0) = true
        else
            Listbox1.CellCheck(i,0) = false
        end
    next
End Sub

```

### **bitmask8\_editor\_control\_RWS.update\_value:**

```

Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 8)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
    for i as integer = 0 to Listbox1.ListCount - 1
        if Listbox1.CellCheck(i,0) then
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        else
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        end
    next

    value = val("&b" + temp_str)
End Sub

```

bitmask8\_editor\_control\_RWS.write:

```
Sub write()  
    update_value  
  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset + reflexive_offset  
    bw.WriteInt8(value)  
    bw.Close  
End Sub  
fs As folderItem
```

**bitmask8\_editor\_control\_RWS.labels():**

```
labels() As string  
//the label  
offset As Integer
```

reflexive\_offset As Integer

value As int8

**bitmask8\_editor\_control\_RWS.values():**

```
values() As Integer  
//the i-th bit
```

**bitmask8\_editor\_control\_RWS Control ListBox1:**

```
Sub CellAction(row As Integer, column As Integer)  
    //this is where to check to see if they checked a box  
    update_value  
    write  
End Sub  
End Class
```

## **Class bitmask16\_editor\_control\_RWS**

Inherits ContainerControl

**bitmask16\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()  
    me.read  
End Sub
```

**bitmask16\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as integer)  
    fs = f  
    offset = in_offset  
    reflexive_offset = 0  
    labels = in_labels  
    values = in_values  
End Sub
```

### **bitmask16\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt16
    br.Close

    Listbox1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        Listbox1.AddRow(labels(i))
        ListBox1.CellTag(i,0) = i
    next
    Listbox1.ColumnType(0) = 2

    dim temp_str as string = bin(value)
    while temp_str.len < 16
        temp_str = "0" + temp_str
    wend
    for i as integer = 0 to Listbox1.ListCount - 1
        if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
            Listbox1.CellCheck(i,0) = true
        else
            Listbox1.CellCheck(i,0) = false
        end
    next
End Sub
```

### **bitmask16\_editor\_control\_RWS.update\_value:**

```
Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 16)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
    for i as integer = 0 to Listbox1.ListCount - 1
        if ListBox1.CellCheck(i,0) then
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        else
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        end
    next

    value = val("&b" + temp_str)
End Sub
```

### **bitmask16\_editor\_control\_RWS.write:**

```
Sub write()
    update_value
End Sub
```

```

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt16(value)
    bw.Close
End Sub
fs As folderItem

```

### **bitmask16\_editor\_control\_RWS.labels():**

```

labels() As string
//the label
offset As Integer

```

```

reflexive_offset As Integer

```

```

value As int16

```

### **bitmask16\_editor\_control\_RWS.values():**

```

values() As Integer
//the i-th bit

```

### **bitmask16\_editor\_control\_RWS Control ListBox1:**

```

Sub CellAction(row As Integer, column As Integer)
    //this is where to check to see if they checked a box
    update_value
    write
End Sub
End Class

```

## **Class bitmask32\_editor\_control\_RWS**

Inherits ContainerControl

### **bitmask32\_editor\_control\_RWS.LostFocus:**

```

Sub LostFocus()
    me.read
End Sub

```

### **bitmask32\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    labels = in_labels
    values = in_values
End Sub

```

### **bitmask32\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile

```

```

br.LittleEndian = true
br.Position = offset + reflexive_offset
value = br.ReadInt32
br.Close

Listbox1.DeleteAllRows
for i as integer = 0 to UBound(labels)
    Listbox1.AddRow(labels(i))
    ListBox1.CellTag(i,0) = i
next
Listbox1.ColumnType(0) = 2

dim temp_str as string = bin(value)
while temp_str.len < 32
    temp_str = "0" + temp_str
wend
for i as integer = 0 to Listbox1.ListCount - 1
    if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
        Listbox1.CellCheck(i,0) = true
    else
        Listbox1.CellCheck(i,0) = false
    end
next
End Sub

```

### **bitmask32\_editor\_control\_RWS.update\_value:**

```

Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 32)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
    for i as integer = 0 to Listbox1.ListCount - 1
        if ListBox1.CellCheck(i,0) then
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        else
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        end
    next

    value = val("&b" + temp_str)
End Sub

```

### **bitmask32\_editor\_control\_RWS.write:**

```

Sub write()
    update_value

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt32(value)

```

```
    bw.Close
End Sub
fs As folderItem
```

### **bitmask32\_editor\_control\_RWS.labels():**

```
labels() As string
//the label
offset As Integer
```

```
reflexive_offset As Integer
```

```
value As int32
```

### **bitmask32\_editor\_control\_RWS.values():**

```
values() As Integer
//the i-th bit
```

### **bitmask32\_editor\_control\_RWS Control ListBox1:**

```
Sub CellAction(row As Integer, column As Integer)
    //this is where to check to see if they checked a box
    update_value
    write
End Sub
End Class
```

## **Class colorARGB\_editor\_control\_RWS**

Inherits ContainerControl

### **colorARGB\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    change_ok = false
End Sub
```

### **colorARGB\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset

    alpha = br.ReadSingle
    red = br.ReadSingle
    green = br.ReadSingle
    blue = br.ReadSingle

    br.Close
```

```

    Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
    change_ok = false
    Slider1.Value = alpha*100
    EditField1.Text = str(Slider1.Value)
    change_ok = true
End Sub

```

#### **colorARGB\_editor\_control\_RWS.update\_value:**

```

Sub update_value()
    alpha = Slider1.Value/100.0
    EditField1.Text = str(Slider1.Value)
    Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
End Sub

```

#### **colorARGB\_editor\_control\_RWS.write:**

```

Sub write()
    update_value
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = Offset + reflexive_offset
    bw.WriteSingle(alpha)
    bw.WriteSingle(red)
    bw.WriteSingle(green)
    bw.WriteSingle(blue)
    bw.close
End Sub

```

alpha As single

blue As single

change\_ok As boolean

fs As folderItem

green As single

offset As Integer

red As single

reflexive\_offset As Integer

#### **colorARGB\_editor\_control\_RWS Control BevelButton1:**

```

Sub Action()
    if not change_ok then return
    dim temp_color as color = RGB(red * 255, green * 255, blue * 255)

    if SelectColor(temp_color, "Select a Color") then
        dim R as single = temp_color.Red
        dim G as single = temp_color.Green
        dim B as single = temp_color.Blue
        red = R/255.0
    end if
End Sub

```

```

        green = G/255.0
        blue = B/255.0
        update_value
        write
    end
End Sub
colorARGB_editor_control_RWS Control Slider1:

```

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub
End Class

```

## **Class colorbyte\_editor\_control\_RWS**

Inherits ContainerControl

### **colorbyte\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    change_ok = false
End Sub

```

### **colorbyte\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset

    blue = br.ReadUInt8
    green = br.ReadUInt8
    red = br.ReadUInt8
    alpha = br.ReadUInt8

    br.Close

    Rectangle1.FillColor = RGB(red, green, blue)
    change_ok = false
    Slider1.Value = (alpha/255.0)*100
    EditField1.Text = str(Slider1.Value)
    change_ok = true
End Sub

```

### **colorbyte\_editor\_control\_RWS.update\_value:**

```

Sub update_value()
    alpha = (Slider1.Value/100.0) * 255
    EditField1.Text = str(Slider1.Value)
    Rectangle1.FillColor = RGB(red, green, blue)
End Sub

```



### **colorbyte\_editor\_control\_RWS.write:**

```
Sub write()  
    update_value  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = Offset + reflexive_offset  
    bw.Writeuint8(blue)  
    bw.Writeuint8(green)  
    bw.Writeuint8(red)  
    bw.Writeuint8(alpha)  
    bw.close  
End Sub  
alpha As uint8  
  
blue As uint8  
  
change_ok As boolean  
  
fs As folderItem  
  
green As uint8  
  
offset As Integer  
  
red As uint8  
  
reflexive_offset As Integer
```

### **colorbyte\_editor\_control\_RWS Control BevelButton1:**

```
Sub Action()  
    if not change_ok then return  
    dim temp_color as color = RGB(red, green, blue)  
  
    if SelectColor(temp_color, "Select a Color") then  
        red = temp_color.Red  
        green = temp_color.Green  
        blue = temp_color.Blue  
        update_value  
        write  
    end  
End Sub
```

### **colorbyte\_editor\_control\_RWS Control Slider1:**

```
Sub ValueChanged()  
    if not change_ok then return  
    update_value  
    write  
End Sub  
End Class
```

## **Class colorRGB\_editor\_control\_RWS**

Inherits ContainerControl

### **colorRGB\_editor\_control\_RWS.Constructor:**

Sub Constructor(f as folderItem, in\_offset as integer)

```
fs = f
offset = in_offset
reflexive_offset = 0
change_ok = false
```

End Sub

### **colorRGB\_editor\_control\_RWS.read:**

Sub read()

```
dim br as BinaryStream = fs.OpenAsBinaryFile
br.LittleEndian = true
br.Position = offset + reflexive_offset
red = br.ReadSingle
green = br.ReadSingle
blue = br.ReadSingle

br.Close

Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
change_ok = true
```

End Sub

### **colorRGB\_editor\_control\_RWS.update\_value:**

Sub update\_value()

```
Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
```

End Sub

### **colorRGB\_editor\_control\_RWS.write:**

Sub write()

```
update_value
dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = Offset + reflexive_offset
bw.WriteSingle(red)
bw.WriteSingle(green)
bw.WriteSingle(blue)
bw.close
```

End Sub

blue As single

change\_ok As boolean

fs As folderItem

green As single

offset As Integer

red As single

reflexive\_offset As Integer

### **colorRGB\_editor\_control\_RWS Control BevelButton1:**

```
Sub Action()  
    if not change_ok then return  
    dim temp_color as color = RGB(red * 255, green * 255, blue * 255)  
  
    if SelectColor(temp_color, "Select a Color") then  
        dim R as single = temp_color.Red  
        dim G as single = temp_color.Green  
        dim B as single = temp_color.Blue  
        red = R/255.0  
        green = G/255.0  
        blue = B/255.0  
        update_value  
        write  
    end  
End Sub  
End Class
```

### **Class dependency\_editor\_control\_RWS**

Inherits ContainerControl

#### **dependency\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()  
    me.read  
End Sub
```

#### **dependency\_editor\_control\_RWS.class\_changed:**

```
Sub class_changed()  
    if change_ok then  
        dim class_index as integer = PopupMenu1.ListIndex  
        change_ok = false  
        PopupMenu2.DeleteAllRows  
        PopupMenu2.AddRow("nulled out")  
        PopupMenu2.RowTag(0) = &hFFFFFFF  
        dim indexs(-1) as integer  
        dim strings(-1) as string  
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)  
            dim index as integer = map.meta_class_folders.child(class_index).item(i)  
            indexs.append index  
            strings.append map.tags(index).name  
        next  
        strings.sortwith(indexs)  
        for i as integer = 0 to UBound(indexs)  
            PopupMenu2.AddRow(map.tags(indexs(i)).name)  
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID  
        next  
        PopupMenu2.ListIndex = -1  
        change_ok = true  
    end if  
End Sub
```

```
end
End Sub
```

### **dependency\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, byref in_map as H1.map)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    map = in_map
    change_ok = false
End Sub
```

### **dependency\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value_class_str = br.Read(4)
    value_name_offset = br.ReadInt32 - map.magic
    br.Position = br.Position + 4
    value_ID = br.ReadUInt32
    if value_ID <> &hFFFFFFFF then
        if map.tagIDtable.haskey(value_ID) then
            value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
            value_name_offset = map.tags(map.tagIDtable.value(value_ID)).nameoffset
        end
    end
    br.Close

    update
End Sub
```

### **dependency\_editor\_control\_RWS.update:**

```
Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    dim class_index as integer = -1
    for i as integer = 0 to UBound(map.meta_class_folders.child)
        dim temp_str() as string = split(map.meta_class_folders.child(i).name, ":")
        dim temp_class1_str as string = temp_str(0)
        PopupMenu1.AddRow(temp_class1_str)
        if value_class_str = reverse(temp_class1_str) then
            PopupMenu1.ListIndex = i
            class_index = i
        end
    next
    if class_index = -1 then
        PopupMenu1.ListIndex = -1
    end
    PopupMenu2.DeleteAllRows
    PopupMenu2.AddRow("nulled out")
    PopupMenu2.RowTag(0) = &hFFFFFFFF
    if value_ID = &hFFFFFFFF then
```

```

        PopupMenu2.ListIndex = 0
    end
    if class_index > -1 then
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
            if value_ID = map.tags(indexs(i)).tagID then
                PopupMenu2.ListIndex = i+1
            end
        next
    end
    change_ok = true
    me.Refresh
End Sub

```

### **dependency\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.Write(value_class_str)
    bw.WriteInt32(value_name_offset + map.magic)
    bw.WriteInt32(0)
    bw.WriteUInt32(value_ID)
    bw.Close
End Sub

```

change\_ok As boolean

fs As folderItem

map As H1.map

offset As Integer

reflexive\_offset As Integer

value\_class\_str As string

value\_ID As uint32

value\_name\_offset As Integer

### **dependency\_editor\_control\_RWS Control PopupMenu1:**

```

Sub Change()
    if change_ok then

```

```

        class_changed
    end
End Sub

dependency_editor_control_RWS Control PopupMenu2:

Sub Change()
    if change_ok then
        value_ID = PopupMenu2.RowTag(PopupMenu2.ListIndex)
        if value_ID <> &hFFFFFF then
            if map.tagIDtable.haskey(value_ID) then
                value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
                value_name_offset = map.tags(map.tagIDtable.value(value_ID)).nameoffset
            end
        end
    end
    write
end
End Sub
End Class

```

## **Class double\_editor\_control\_RWS**

Inherits ContainerControl

### **double\_editor\_control\_RWS.LostFocus:**

```

Sub LostFocus()
    me.read
End Sub

```

### **double\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub

```

### **double\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.Readdouble
    EditField1.Text = str(value)
    br.Close
End Sub

```

### **double\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.Writedouble(value)
    bw.Close

```

```
End Sub
fs As folderItem
```

```
offset As Integer
```

```
reflexive_offset As Integer
```

```
value As double
```

### **double\_editor\_control\_RWS Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class float\_editor\_control\_RWS**

Inherits ContainerControl

### **float\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()
    me.read
End Sub
```

### **float\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub
```

### **float\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadSingle
    EditField1.Text = str(value)
    br.Close
End Sub
```

### **float\_editor\_control\_RWS.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.Writesingle(value)
```

```
    bw.Close
End Sub
fs As folderItem
```

```
offset As Integer
```

```
reflexive_offset As Integer
```

```
value As single
```

### **float\_editor\_control\_RWS Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class id8\_editor\_control\_RWS**

Inherits ContainerControl

### **id8\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as int8)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    labels = in_labels
    values = in_values
    change_ok = false
End Sub
```

### **id8\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt8
    br.Close

    update
End Sub
```

### **id8\_editor\_control\_RWS.update:**

```
Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        PopupMenu1.AddRow(labels(i))
    next i
End Sub
```



```

        if value = values(i) then
            PopupMenu1.ListIndex = i
        end
    next
    change_ok = true
End Sub

```

### **id8\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt8(value)
    bw.Close
End Sub
change_ok As boolean

```

fs As folderItem

labels() As string

offset As Integer

reflexive\_offset As Integer

value As int8

values() As int8

### **id8\_editor\_control\_RWS Control PopupMenu1:**

```

Sub Change()
    if change_ok then
        value = values(me.ListIndex)
        write
    end
End Sub
End Class

```

## **Class id16\_editor\_control\_RWS**

Inherits ContainerControl

### **id16\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as int16)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    labels = in_labels
    values = in_values
    change_ok = false
End Sub

```

**id16\_editor\_control\_RWS.read:**

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset + reflexive_offset  
    value = br.ReadInt16  
    br.Close  
  
    update  
End Sub
```

**id16\_editor\_control\_RWS.update:**

```
Sub update()  
    change_ok = false  
    PopupMenu1.DeleteAllRows  
    for i as integer = 0 to UBound(labels)  
        PopupMenu1.AddRow(labels(i))  
        if value = values(i) then  
            PopupMenu1.ListIndex = i  
        end  
    next  
    change_ok = true  
End Sub
```

**id16\_editor\_control\_RWS.write:**

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset + reflexive_offset  
    bw.WriteInt16(value)  
    bw.Close  
End Sub  
change_ok As boolean
```

fs As folderItem

labels() As string

offset As Integer

reflexive\_offset As Integer

value As int16

values() As int16

**id16\_editor\_control\_RWS Control PopupMenu1:**

```
Sub Change()  
    if change_ok then  
        value = values(me.ListIndex)  
        write  
    end if  
End Sub
```

```
end
End Sub
End Class
```

## **Class id32\_editor\_control\_RWS**

Inherits ContainerControl

### **id32\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
    labels = in_labels
    values = in_values
    change_ok = false
End Sub
```

### **id32\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt32
    br.Close

    update

End Sub
```

### **id32\_editor\_control\_RWS.update:**

```
Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        PopupMenu1.AddRow(labels(i))
        if value = values(i) then
            PopupMenu1.ListIndex = i
        end
    next
    change_ok = true
End Sub
```

### **id32\_editor\_control\_RWS.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt32(value)
    bw.Close
End Sub
change_ok As boolean
```

fs As folderItem

labels() As string

offset As Integer

reflexive\_offset As Integer

value As int32

values() As int32

### **id32\_editor\_control\_RWS Control PopupMenu1:**

Sub Change()

    if change\_ok then

        value = values(me.ListIndex)

        write

    end

End Sub

End Class

## **Class index\_editor\_control\_RWS**

Inherits ContainerControl

### **index\_editor\_control\_RWS.Constructor:**

Sub Constructor(f as folderItem, in\_offset as integer, in\_path as string)

    fs = f

    offset = in\_offset

    reflexive\_offset = 0

    change\_ok = false

    path = in\_path

    path\_text.text = path.ReplaceAll(":", ": ")

    //something extra

End Sub

### **index\_editor\_control\_RWS.read:**

Sub read()

    max\_count = reflexive\_pointed\_at.chunk\_count

    dim br as BinaryStream = fs.OpenAsBinaryFile

    br.LittleEndian = true

    br.Position = offset + reflexive\_offset

    value = br.ReadInt16

    br.Close

    update

End Sub

### **index\_editor\_control\_RWS.update:**

Sub update()

    change\_ok = false

```

PopupMenu1.DeleteAllRows
PopupMenu1.AddRow("None Selected")
dim br as BinaryStream = reflexive_pointed_at.fs.OpenAsBinaryFile
br.littleendian = true
br.position = reflexive_pointed_at.offset + reflexive_pointed_at.reflexive_offset
max_count = br.readint32 - 1
dim chunk_names(-1) as string
if reflexive_pointed_at.name_offset <> -1 then
    for i as integer = 0 to max_count
        br.position = reflexive_pointed_at.chunk_offset + ((i)*reflexive_pointed_at.chunk_size)_
            + reflexive_pointed_at.name_offset
        dim temp_str as string = br.read(32)
        temp_str = temp_str.replaceAll(chr(0),"")
        chunk_names.append temp_str
    next
end
if reflexive_pointed_at.name_offset <> -1 _
    and max_count <> 0 then
    for i as integer = 0 to max_count
        PopupMenu1.AddRow(chunk_names(i))
    next
else
    for i as integer = 0 to max_count
        PopupMenu1.AddRow("Chunk " + str(i+1))
    next
end
PopupMenu1.ListIndex = value + 1
change_ok = true
br.Close
End Sub

```

### **index\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt16(value)
    bw.Close
End Sub
change_ok As boolean

```

fs As folderItem

max\_count As Integer

offset As Integer

path As string

reflexive\_offset As Integer

reflexive\_pointed\_at As reflexive\_control\_RWS

value As int16

## **index\_editor\_control\_RWS Control PopupMenu1:**

```
Sub Change()  
    if change_ok then  
        value = me.ListIndex - 1  
        write  
    end  
End Sub  
End Class
```

## **Class int8\_editor\_control\_RWS**

Inherits ContainerControl

### **int8\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()  
    me.read  
End Sub
```

### **int8\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer)  
    fs = f  
    offset = in_offset  
    reflexive_offset = 0  
End Sub
```

### **int8\_editor\_control\_RWS.read:**

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset + reflexive_offset  
    value = br.ReadInt8  
    EditField1.Text = str(value)  
    br.Close  
End Sub
```

### **int8\_editor\_control\_RWS.write:**

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset + reflexive_offset  
    bw.WriteInt8(value)  
    bw.Close  
End Sub  
fs As folderItem
```

offset As Integer

reflexive\_offset As Integer

value As Int8

## **int8\_editor\_control\_RWS Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class int16\_editor\_control\_RWS**

Inherits ContainerControl

### **int16\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()
    me.read
End Sub
```

### **int16\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub
```

### **int16\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt16
    EditField1.Text = str(value)
    br.Close
End Sub
```

### **int16\_editor\_control\_RWS.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt16(value)
    bw.Close
End Sub
fs As folderItem
```

offset As Integer

reflexive\_offset As Integer

value As Int16

### **int16\_editor\_control\_RWS Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class int32\_editor\_control\_RWS**

Inherits ContainerControl

### **int32\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()
    me.read
End Sub
```

### **int32\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub
```

### **int32\_editor\_control\_RWS.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.ReadInt32
    EditField1.Text = str(value)
    br.Close
End Sub
```

### **int32\_editor\_control\_RWS.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteInt32(value)
    bw.Close
End Sub
fs As folderItem
```

offset As Integer

reflexive\_offset As Integer



value As Int32

### **int32\_editor\_control\_RWS Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class loneID\_editor\_control\_RWS**

Inherits ContainerControl

### **loneID\_editor\_control\_RWS.LostFocus:**

```
Sub LostFocus()
    me.read
End Sub
```

### **loneID\_editor\_control\_RWS.class\_changed:**

```
Sub class_changed()
    if change_ok then
        dim class_index as integer = PopupMenu1.ListIndex
        change_ok = false
        PopupMenu2.DeleteAllRows
        PopupMenu2.AddRow("nulled out")
        PopupMenu2.RowTag(0) = &hFFFFFFF
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
        next
        PopupMenu2.ListIndex = -1
        change_ok = true
    end
End Sub
```

### **loneID\_editor\_control\_RWS.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, byref in_map as H1.map)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub
```

```

map = in_map
change_ok = false
End Sub

```

### **loneID\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value_ID = br.ReadUInt32
    value_class_str = "none"
    if value_ID <> &hFFFFFFFF then
        if map.tagIDtable.haskey(value_ID) then
            value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
        end
    end
    br.Close

    update
End Sub

```

### **loneID\_editor\_control\_RWS.update:**

```

Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    dim class_index as integer = -1
    for i as integer = 0 to UBound(map.meta_class_folders.child)
        dim temp_str() as string = split(map.meta_class_folders.child(i).name, ":")
        dim temp_class1_str as string = temp_str(0)
        PopupMenu1.AddRow(temp_class1_str)
        if value_class_str = reverse(temp_class1_str) then
            PopupMenu1.ListIndex = i
            class_index = i
        end
    next
    if class_index = -1 then
        PopupMenu1.ListIndex = -1
    end
    PopupMenu2.DeleteAllRows
    PopupMenu2.AddRow("nulled out")
    PopupMenu2.RowTag(0) = &hFFFFFFFF
    if value_ID = &hFFFFFFFF then
        PopupMenu2.ListIndex = 0
    end
    if class_index > -1 then
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)

```

```

        PopupMenu2.AddRow(map.tags(indexs(i)).name)
        PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
        if value_ID = map.tags(indexs(i)).tagID then
            PopupMenu2.ListIndex = i+1
        end
    next
end
change_ok = true
me.Refresh
End Sub

```

### **loneID\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.WriteUInt32(value_ID)
    bw.Close
End Sub
change_ok As boolean

```

fs As folderItem

map As H1.map

offset As Integer

reflexive\_offset As Integer

value\_class\_str As string

value\_ID As uint32

### **loneID\_editor\_control\_RWS Control PopupMenu1:**

```

Sub Change()
    if change_ok then
        class_changed
    end
End Sub

```

### **loneID\_editor\_control\_RWS Control PopupMenu2:**

```

Sub Change()
    if change_ok then
        value_ID = PopupMenu2.RowTag(PopupMenu2.ListIndex)
        if value_ID <> &hFFFFFFFF then
            if map.tagIDtable.haskey(value_ID) then
                value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
            end
        end
        write
    end
End Sub

```

End Class

## **Class string4\_editor\_control\_RWS**

Inherits ContainerControl

### **string4\_editor\_control\_RWS.LostFocus:**

Sub LostFocus()

    me.read

End Sub

### **string4\_editor\_control\_RWS.Constructor:**

Sub Constructor(f as folderItem, in\_offset as integer)

    fs = f

    offset = in\_offset

    reflexive\_offset = 0

End Sub

### **string4\_editor\_control\_RWS.read:**

Sub read()

    dim br as BinaryStream = fs.OpenAsBinaryFile

    br.LittleEndian = true

    br.Position = offset + reflexive\_offset

    value = br.Read(4)

    EditField1.Text = value

    br.Close

End Sub

### **string4\_editor\_control\_RWS.write:**

Sub write()

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)

    bw.LittleEndian = true

    bw.Position = offset + reflexive\_offset

    bw.Write(value)

    bw.Close

End Sub

fs As folderItem

offset As Integer

reflexive\_offset As Integer

value As string

### **string4\_editor\_control\_RWS Control EditField1:**

Function KeyDown(Key As String) As Boolean

    if asc(key) = 3 or asc(key) = 13 then

        value = EditField1.text

        value = value.LeftB(4)

        while value.LenB < 4

            value = value + chr(0)

```

        wend
        EditField1.Text = value
        write
        return true
    end
End Function
string4_editor_control_RWS Control PushButton1:

```

```

Sub Action()
    EditField1.text = EditField1.Text + chr(0)
End Sub
End Class

```

## **Class string32\_editor\_control\_RWS**

Inherits ContainerControl

### **string32\_editor\_control\_RWS.LostFocus:**

```

Sub LostFocus()
    me.read
End Sub

```

### **string32\_editor\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer)
    fs = f
    offset = in_offset
    reflexive_offset = 0
End Sub

```

### **string32\_editor\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    value = br.Read(32)
    EditField1.Text = value
    br.Close
End Sub

```

### **string32\_editor\_control\_RWS.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.Write(value)
    bw.Close
End Sub
fs As folderItem

```

offset As Integer

reflexive\_offset As Integer

value As string

### **string32\_editor\_control\_RWS Control EditField1:**

Function KeyDown(Key As String) As Boolean

if asc(key) = 3 or asc(key) = 13 then

value = EditField1.text

value = value.LeftB(32)

while value.LenB < 32

value = value + chr(0)

wend

EditField1.Text = value

write

return true

end

End Function

### **string32\_editor\_control\_RWS Control PushButton1:**

Sub Action()

EditField1.text = EditField1.Text + chr(0)

End Sub

End Class

## **Class string128\_editor\_control\_RWS**

Inherits ContainerControl

### **string128\_editor\_control\_RWS.LostFocus:**

Sub LostFocus()

me.read

End Sub

### **string128\_editor\_control\_RWS.Constructor:**

Sub Constructor(f as folderItem, in\_offset as integer)

fs = f

offset = in\_offset

reflexive\_offset = 0

End Sub

### **string128\_editor\_control\_RWS.read:**

Sub read()

dim br as BinaryStream = fs.OpenAsBinaryFile

br.LittleEndian = true

br.Position = offset + reflexive\_offset

value = br.Read(128)

EditField1.Text = value

br.Close

End Sub

### **string128\_editor\_control\_RWS.write:**

Sub write()

```

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset + reflexive_offset
    bw.Write(value)
    bw.Close
End Sub
fs As folderItem

```

```
offset As Integer
```

```
reflexive_offset As Integer
```

```
value As string
```

### **string128\_editor\_control\_RWS Control EditField1:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = EditField1.text
        value = value.LeftB(128)
        while value.LenB < 128
            value = value + chr(0)
        wend
        EditField1.Text = value
        write
        return true
    end
End Function

```

### **string128\_editor\_control\_RWS Control PushButton1:**

```

Sub Action()
    EditField1.text = EditField1.Text + chr(0)
End Sub
End Class

```

## **Class reflexive\_control\_RWS**

Inherits ContainerControl

### **reflexive\_control\_RWS.add\_granddaddy:**

```

Sub add_granddaddy(byref c as main_control_RWS)
    grand_parent = c
End Sub

```

### **reflexive\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_chunk_size as integer, in_magic as integer, in_name_offset as integer, in_reflexive_offset as integer = 0)
    fs = f
    offset = in_offset
    chunk_size = in_chunk_size
    magic = in_magic
    reflexive_offset = in_reflexive_offset
    change_ok = false

```

```
name_offset = in_name_offset
End Sub
```

### **reflexive\_control\_RWS.Destructor:**

```
Sub Destructor()
    while UBound(bitmask16s) > -1
        bitmask16s(0).close
        bitmask16s.Remove(0)
    wend
    while UBound(bitmask32s) > -1
        bitmask32s(0).close
        bitmask32s.Remove(0)
    wend
    while UBound(bitmask16s) > -1
        bitmask8s(0).close
        bitmask8s.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorARGBs(0).Close
        colorARGBs.Remove(0)
    wend
    while UBound(colorbytes) > -1
        colorbytes(0).Close
        colorbytes.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorRGBs(0).Close
        colorRGBs.Remove(0)
    wend
    While UBound(dependencys) > -1
        dependencys(0).Close
        dependencys.Remove(0)
    wend
    while UBound(doubles) > -1
        doubles(0).close
        doubles.Remove(0)
    wend
    while UBound(floats) > -1
        floats(0).close
        floats.Remove(0)
    wend
    while UBound(id16s) > -1
        id16s(0).close
        id16s.Remove(0)
    wend
    while UBound(id32s) > -1
        id32s(0).close
        id32s.Remove(0)
    wend
    while UBound(id8s) > -1
        id8s(0).close
        id8s.Remove(0)
    wend
    while UBound(indexs) > -1
```



```

        indexs(0).close
        indexs.Remove(0)
    wend
    while UBound(int16s) > -1
        int16s(0).close
        int16s.Remove(0)
    wend
    while UBound(int32s) > -1
        int32s(0).close
        int32s.Remove(0)
    wend
    while UBound(int8s) > -1
        int8s(0).close
        int8s.Remove(0)
    wend
    while UBound(loneIDs) > -1
        loneIDs(0).close
        loneIDs.Remove(0)
    wend
    while UBound(reflexives) > -1
        reflexives(0).close
        reflexives.Remove(0)
    wend
    while UBound(string128s) > -1
        string128s(0).close
        string128s.Remove(0)
    wend
    while UBound(string32s) > -1
        string32s(0).close
        string32s.Remove(0)
    wend
    while UBound(string4s) > -1
        string4s(0).close
        String4s.Remove(0)
    wend
End Sub

```

### **reflexive\_control\_RWS.disable\_controls:**

```

Sub disable_controls()
    for i as integer = 0 to UBound(bitmask16s)
        bitmask16s(i).enabled = false
    next
    for i as integer = 0 to UBound(bitmask32s)
        bitmask32s(i).enabled = false
    next
    for i as integer = 0 to UBound(bitmask8s)
        bitmask8s(i).enabled = false
    next
    for i as integer = 0 to UBound(colorARGBs)
        colorARGBs(i).enabled = false
    next
    for i as integer = 0 to UBound(colorbytes)
        colorbytes(i).enabled = false
    next

```

```

for i as integer = 0 to UBound(colorRGBs)
    colorRGBs(i).enabled = false
next
for i as integer = 0 to UBound(dependencys)
    dependencys(i).Enabled = false
next
for i as integer = 0 to UBound(doubles)
    doubles(i).enabled = false
next
for i as integer = 0 to UBound(floats)
    floats(i).enabled = false
next
for i as integer = 0 to UBound(id16s)
    id16s(i).enabled = false
next
for i as integer = 0 to UBound(id32s)
    id32s(i).enabled = false
next
for i as integer = 0 to UBound(id8s)
    id8s(i).enabled = false
next
for i as integer = 0 to UBound(indexs)
    indexs(i).enabled = false
next
for i as integer = 0 to UBound(int16s)
    int16s(i).enabled = false
next
for i as integer = 0 to UBound(int32s)
    int32s(i).enabled = false
next
for i as integer = 0 to UBound(int8s)
    int8s(i).enabled = false
next
for i as integer = 0 to UBound(loneIDs)
    loneIDs(i).Enabled = false
next
for i as integer = 0 to UBound(reflexives)
    reflexives(i).enabled = false
    reflexives(i).chunk_count = 0
next
for i as integer = 0 to UBound(string128s)
    string128s(i).enabled = false
next
for i as integer = 0 to UBound(string32s)
    string32s(i).enabled = false
next
for i as integer = 0 to UBound(string4s)
    string4s(i).enabled = false
next
End Sub

```

### **reflexive\_control\_RWS.init:**

```

Sub init(plugin_info as pluginLibUniversal, edit_by_offset as boolean, byref map as H1.map)
    redim bitmask16s(-1)

```

```

redim bitmask32s(-1)
redim bitmask8s(-1)
redim colorARGBs(-1)
redim colorbytes(-1)
redim colorRGBs(-1)
redim dependencys(-1)
redim doubles(-1)
redim floats(-1)
redim id16s(-1)
redim id32s(-1)
redim id8s(-1)
redim indexs(-1)
redim int16s(-1)
redim int32s(-1)
redim int8s(-1)
redim lonelDs(-1)
redim reflexives(-1)
redim string128s(-1)
redim string32s(-1)
redim string4s(-1)

```

```
//so organize the various editing components of the main reflexive
```

```

dim index_list(-1) as string
dim offset_list(-1) as integer
dim order_list(-1) as integer
for i as integer = 0 to UBound(plugin_info.bitmask16_offset)
    index_list.Append("bitmask16:" + str(i))
    offset_list.Append plugin_info.bitmask16_offset(i)
    order_list.Append plugin_info.bitmask16_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask32_offset)
    index_list.Append("bitmask32:" + str(i))
    offset_list.Append plugin_info.bitmask32_offset(i)
    order_list.Append plugin_info.bitmask32_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask8_offset)
    index_list.Append("bitmask8:" + str(i))
    offset_list.Append plugin_info.bitmask8_offset(i)
    order_list.Append plugin_info.bitmask8_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorARGB_offset)
    index_list.Append("colorARGB:" + str(i))
    offset_list.Append plugin_info.colorARGB_offset(i)
    order_list.Append plugin_info.colorARGB_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorbyte_offset)
    index_list.Append("colorbyte:" + str(i))
    offset_list.Append plugin_info.colorbyte_offset(i)
    order_list.Append plugin_info.colorbyte_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorRGB_offset)
    index_list.Append("colorRGB:" + str(i))
    offset_list.Append plugin_info.colorRGB_offset(i)
    order_list.Append plugin_info.colorRGB_order(i)

```

```

next
for i as integer = 0 to UBound(plugin_info.dependency_offset)
    index_list.Append("dependency:" + str(i))
    offset_list.Append plugin_info.dependency_offset(i)
    order_list.Append plugin_info.dependency_order(i)
next
for i as integer = 0 to UBound(plugin_info.double_offset)
    index_list.Append("double:" + str(i))
    offset_list.Append plugin_info.double_offset(i)
    order_list.Append plugin_info.double_order(i)
next
for i as integer = 0 to UBound(plugin_info.float_offset)
    index_list.Append("float:" + str(i))
    offset_list.Append plugin_info.float_offset(i)
    order_list.Append plugin_info.float_order(i)
next
for i as integer = 0 to UBound(plugin_info.id16_offset)
    index_list.Append("id16:" + str(i))
    offset_list.Append plugin_info.id16_offset(i)
    order_list.Append plugin_info.id16_order(i)
next
for i as integer = 0 to UBound(plugin_info.id32_offset)
    index_list.Append("id32:" + str(i))
    offset_list.Append plugin_info.id32_offset(i)
    order_list.Append plugin_info.id32_order(i)
next
for i as integer = 0 to UBound(plugin_info.id8_offset)
    index_list.Append("id8:" + str(i))
    offset_list.Append plugin_info.id8_offset(i)
    order_list.Append plugin_info.id8_order(i)
next
for i as integer = 0 to UBound(plugin_info.index_offset)
    index_list.Append("index:" + str(i))
    offset_list.Append plugin_info.index_offset(i)
    order_list.Append plugin_info.index_order(i)
next
for i as integer = 0 to UBound(plugin_info.int16_offset)
    index_list.Append("int16:" + str(i))
    offset_list.Append plugin_info.int16_offset(i)
    order_list.Append plugin_info.int16_order(i)
next
for i as integer = 0 to UBound(plugin_info.int32_offset)
    index_list.Append("int32:" + str(i))
    offset_list.Append plugin_info.int32_offset(i)
    order_list.Append plugin_info.int32_order(i)
next
for i as integer = 0 to UBound(plugin_info.int8_offset)
    index_list.Append("int8:" + str(i))
    offset_list.Append plugin_info.int8_offset(i)
    order_list.Append plugin_info.int8_order(i)
next
for i as integer = 0 to UBound(plugin_info.loneID_offset)
    index_list.Append("loneID:" + str(i))
    offset_list.Append plugin_info.loneID_offset(i)

```

```

    order_list.Append plugin_info.lonelD_order(i)
next
for i as Integer = 0 to UBound(plugin_info.reflexive_offset)
    index_list.Append("reflexive:" + str(i))
    offset_list.Append plugin_info.reflexive_offset(i)
    order_list.Append plugin_info.reflexive_order(i)
next
for i as integer = 0 to UBound(plugin_info.string128_offset)
    index_list.Append("string128:" + str(i))
    offset_list.Append plugin_info.string128_offset(i)
    order_list.Append plugin_info.string128_order(i)
next
for i as integer = 0 to UBound(plugin_info.string32_offset)
    index_list.Append("string32:" + str(i))
    offset_list.Append plugin_info.string32_offset(i)
    order_list.Append plugin_info.string32_order(i)
next
for i as integer = 0 to UBound(plugin_info.string4_offset)
    index_list.Append("string4:" + str(i))
    offset_list.Append plugin_info.string4_offset(i)
    order_list.Append plugin_info.string4_order(i)
next

if edit_by_offset then
    offset_list.sortwith(index_list)
else
    order_list.sortwith(index_list, offset_list)
end

//now create the individual controls and add them to the main window control
dim temp_height as integer = 12 + 52 //52 height of distance from "top" of the groupbox control to bottom chunk
menu
for i as integer = 0 to UBound(offset_list)
    dim temp_split_str() as string = split(index_list(i), ":")
    select case temp_split_str(0)

        case "bitmask16"
            dim temp_ints(-1) as Integer
            for j as integer = 0 to UBound(plugin_info.bitmask16_data(val(temp_split_str(1))).values)
                temp_ints.Append( plugin_info.bitmask16_data(val(temp_split_str(1))).values(j) )
            next
            dim temp_bitmask as new bitmask16_editor_control_RWS(fs, offset_list(i),
            plugin_info.bitmask16_data(val(temp_split_str(1))).labels, temp_ints)
            temp_bitmask.name_text.Text = plugin_info.bitmask16_name(val(temp_split_str(1)))
            temp_bitmask.EmbedWithin(groupbox1, 20, temp_height)
            bitmask16s.Append temp_bitmask
            temp_height = temp_height + 12 + temp_bitmask.Height

        case "bitmask32"
            dim temp_ints(-1) as Integer
            for j as integer = 0 to UBound(plugin_info.bitmask32_data(val(temp_split_str(1))).values)
                temp_ints.Append( plugin_info.bitmask32_data(val(temp_split_str(1))).values(j) )
            next

```

```

dim temp_bitmask as new bitmask32_editor_control_RWS(fs, offset_list(i),
plugin_info.bitmask32_data(val(temp_split_str(1))).labels, temp_ints)
temp_bitmask.name_text.Text = plugin_info.bitmask32_name(val(temp_split_str(1)))
temp_bitmask.EmbedWithin(groupbox1, 20, temp_height)
bitmask32s.Append temp_bitmask
temp_height = temp_height + 12 + temp_bitmask.Height

case "bitmask8"
dim temp_ints(-1) as Integer
for j as integer = 0 to UBound(plugin_info.bitmask8_data(val(temp_split_str(1))).values)
temp_ints.Append( plugin_info.bitmask8_data(val(temp_split_str(1))).values(j) )
next
dim temp_bitmask as new bitmask8_editor_control_RWS(fs, offset_list(i),
plugin_info.bitmask8_data(val(temp_split_str(1))).labels, temp_ints)
temp_bitmask.name_text.Text = plugin_info.bitmask8_name(val(temp_split_str(1)))
temp_bitmask.EmbedWithin(groupbox1, 20, temp_height)
bitmask8s.Append temp_bitmask
temp_height = temp_height + 12 + temp_bitmask.Height

case "colorARGB"
dim temp_color as new colorARGB_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorARGB_name(val(temp_split_str(1)))
temp_color.EmbedWithin(groupbox1, 20, temp_height)
colorARGBs.Append temp_color
temp_height = temp_height + 12 + temp_color.Height

case "colorbyte"
dim temp_color as new colorbyte_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorbyte_name(val(temp_split_str(1)))
temp_color.EmbedWithin(groupbox1, 20, temp_height)
colorbytes.Append temp_color
temp_height = temp_height + 12 + temp_color.Height

case "colorRGB"
dim temp_color as new colorRGB_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorRGB_name(val(temp_split_str(1)))
temp_color.EmbedWithin(groupbox1, 20, temp_height)
colorRGBs.Append temp_color
temp_height = temp_height + 12 + temp_color.Height

case "dependency"
dim temp_dep as new dependency_editor_control_RWS(fs, offset_list(i), map)
temp_dep.name_text.Text = plugin_info.dependency_name(val(temp_split_str(1)))
temp_dep.EmbedWithin(groupbox1, 20, temp_height)
dependencys.Append temp_dep
temp_height = temp_height + 12 + temp_dep.Height

case "double"
dim temp_double as new double_editor_control_RWS(fs, offset_list(i))
temp_double.name_text.Text = plugin_info.double_name(val(temp_split_str(1)))
temp_double.EmbedWithin(groupbox1, 20, temp_height)
doubles.Append temp_double
temp_height = temp_height + 12 + temp_double.Height

```

```

case "float"
    dim temp_float as new float_editor_control_RWS(fs, offset_list(i))
    temp_float.name_text.Text = plugin_info.float_name(val(temp_split_str(1)))
    temp_float.EmbedWithin(groupbox1, 20, temp_height)
    floats.Append temp_float
    temp_height = temp_height + 12 + temp_float.Height

case "id16"
    dim temp_ints(-1) as Int16
    for j as integer = 0 to UBound(plugin_info.id16_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id16_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id16_editor_control_RWS(fs, offset_list(i),
    plugin_info.id16_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id16_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(groupbox1, 20, temp_height)
    id16s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

case "id32"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.id32_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id32_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id32_editor_control_RWS(fs, offset_list(i),
    plugin_info.id32_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id32_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(groupbox1, 20, temp_height)
    id32s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

case "id8"
    dim temp_ints(-1) as Int8
    for j as integer = 0 to UBound(plugin_info.id8_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id8_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id8_editor_control_RWS(fs, offset_list(i),
    plugin_info.id8_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id8_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(groupbox1, 20, temp_height)
    id8s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

case "index"
    dim temp_index as new index_editor_control_RWS(fs, offset_list(i),
    plugin_info.index_data(val(temp_split_str(1))))
    temp_index.name_text.Text = plugin_info.index_name(val(temp_split_str(1)))
    temp_index.EmbedWithin(groupbox1, 20, temp_height)
    indexes.Append temp_index
    temp_height = temp_height + 12 + temp_index.Height

case "int16"
    dim temp_int as new int16_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int16_name(val(temp_split_str(1)))

```

```

temp_int.EmbedWithin(groupbox1, 20, temp_height)
int16s.Append temp_int
temp_height = temp_height + 12 + temp_int.Height

case "int32"
    dim temp_int as new int32_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int32_name(val(temp_split_str(1)))
    temp_int.EmbedWithin(groupbox1, 20, temp_height)
    int32s.Append temp_int
    temp_height = temp_height + 12 + temp_int.Height

case "int8"
    dim temp_int as new int8_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int8_name(val(temp_split_str(1)))
    temp_int.EmbedWithin(groupbox1, 20, temp_height)
    int8s.Append temp_int
    temp_height = temp_height + 12 + temp_int.Height

case "lonelD"
    dim temp_dep as new lonelD_editor_control_RWS(fs, offset_list(i), map)
    temp_dep.name_text.Text = plugin_info.lonelD_name(val(temp_split_str(1)))
    temp_dep.EmbedWithin(groupbox1, 20, temp_height)
    lonelDs.Append temp_dep
    temp_height = temp_height + 12 + temp_dep.Height

case "reflexive"
    dim temp_reflexive as new reflexive_control_RWS(fs, offset_list(i),
    plugin_info.reflexives(val(temp_split_str(1))).this_reflexive_size, magic, _
    plugin_info.reflexive_name_offset(val(temp_split_str(1))), offset+reflexive_offset)
    temp_reflexive.groupbox1.caption = plugin_info.reflexive_name(val(temp_split_str(1)))
    temp_reflexive.EmbedWithin(groupbox1, 20, temp_Height)
    temp_reflexive.add_granddaddy(grand_parent)
    temp_reflexive.init(plugin_info.reflexives(val(temp_split_str(1))), edit_by_offset, map)
    reflexives.Append temp_reflexive
    temp_height = temp_height + 12 + temp_reflexive.Height

case "string128"
    dim temp_string as new string128_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string128_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(groupbox1, 20, temp_height)
    string128s.Append temp_string
    temp_height = temp_height + 12 + temp_string.Height

case "string32"
    dim temp_string as new string32_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string32_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(groupbox1, 20, temp_height)
    string32s.Append temp_string
    temp_height = temp_height + 12 + temp_string.Height

case "string 4"
    dim temp_string as new string4_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string4_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(groupbox1, 20, temp_height)

```



```

        string4s.Append temp_string
        temp_height = temp_height + 12 + temp_string.Height

    end select
next

//so now all of the stuff has been added to the control
temp_height = temp_height +8 // -12 from the last control +20 space it from the bottom
groupbox1.Height = temp_height //hears hoping, since temp_height is calculated starting from the top of it
me.Height = groupbox1.Height
dim temp_width as integer = 400 //the default width of editing controls
for i as integer = 0 to UBound(reflexives)
    if reflexives(i).Width > temp_width then
        temp_width = reflexives(i).Width
    end
next
temp_width = temp_width + 40 //+20 for left and right padding
groupbox1.Width = temp_width
me.Width = temp_width
End Sub

```

### **reflexive\_control\_RWS.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    if br = nil then break
    br.LittleEndian = true
    br.Position = offset + reflexive_offset
    chunk_count = br.ReadInt32
    chunk_offset = br.ReadInt32 - magic
    dim tag_start as integer = reflexive_offset //assumes reflexive data is always after the current chunk
    if chunk_count > 0 AND chunk_offset >= tag_start then
        change_ok = false
        chunk_menu.DeleteAllRows
        if name_offset = -1 then
            for i as integer = 1 to chunk_count
                chunk_menu.AddRow("Chunk " + str(i))
            next
        else
            //find the actual name of each chunk
            for i as integer = 1 to chunk_count
                br.Position = chunk_offset + ((i-1)*chunk_size) + name_offset
                dim temp_str as string = br.read(32)
                temp_str = temp_str.replaceAll(chr(0),"")
                chunk_menu.AddRow(temp_str)
            next
        end
        chunk_menu.ListIndex = 0
        chunk_number = 0
        chunk_menu.Enabled = true
        chunk_text.Enabled = true
        change_ok = true
        update_indexes
        update_reflexive_offset
    else

```

```

change_ok = false
chunk_menu.DeleteAllRows
me.Enabled = false
chunk_menu.Enabled = false
chunk_text.Enabled = false
disable_controls
end

br.close
End Sub

```

### **reflexive\_control\_RWS.update\_indexes:**

```

Sub update_indexes()
//here we go, go straight to the source
for i as integer = 0 to UBound(indexs)
    dim temp_path() as string = split(indexs(i).path,":")
    dim temp_reflexives() as reflexive_control_RWS = grand_parent.reflexives
    for j as integer = 1 to UBound(temp_path) - 1
        dim ref_name as string = temp_path(j)
        for k as integer = 0 to UBound(temp_reflexives)
            if temp_reflexives(k).groupbox1.caption = ref_name then
                temp_reflexives = temp_reflexives(k).reflexives
                exit for k
            end
        next
    next
    for j as integer = 0 to UBound(temp_reflexives)
        if temp_reflexives(j).groupbox1.caption = temp_path(UBound(temp_path)) then
            indexs(i).reflexive_pointed_at = temp_reflexives(j)
            exit for j
        end
    next
next

for i as integer = 0 to UBound(reflexives)
    reflexives(i).update_indexes
next
End Sub

```

### **reflexive\_control\_RWS.update\_reflexive\_offset:**

```

Sub update_reflexive_offset()
//their own "offset" is the local offset, the reflexive_offset is where the reflexive starts
for i as integer = 0 to UBound(bitmask16s)
    bitmask16s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    bitmask16s(i).read
    bitmask16s(i).Enabled = true
next
for i as integer = 0 to UBound(bitmask32s)
    bitmask32s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    bitmask32s(i).read
    bitmask32s(i).Enabled = true
next
for i as integer = 0 to UBound(bitmask8s)
    bitmask8s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size

```

```

    bitmask8s(i).read
    bitmask8s(i).Enabled = true
next
for i as integer = 0 to UBound(colorARGBs)
    colorARGBs(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    colorARGBs(i).read
    colorARGBs(i).Enabled = true
next
for i as integer = 0 to UBound(colorbytes)
    colorbytes(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    colorbytes(i).read
    colorbytes(i).Enabled = true
next
for i as integer = 0 to UBound(colorRGBs)
    colorRGBs(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    colorRGBs(i).read
    colorRGBs(i).Enabled = true
next
for i as integer = 0 to UBound(dependencys)
    dependencys(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    dependencys(i).read
    dependencys(i).Enabled = true
next
for i as integer = 0 to UBound(doubles)
    doubles(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    doubles(i).read
    doubles(i).Enabled = true
next
for i as integer = 0 to UBound(floats)
    floats(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    floats(i).read
    floats(i).Enabled = true
next
for i as integer = 0 to UBound(id16s)
    id16s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    id16s(i).read
    id16s(i).Enabled = true
next
for i as integer = 0 to UBound(id32s)
    id32s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    id32s(i).read
    id32s(i).Enabled = true
next
for i as integer = 0 to UBound(id8s)
    id8s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    id8s(i).read
    id8s(i).Enabled = true
next
//would normally put indexes here but they need to be after reflexives
for i as integer = 0 to UBound(int16s)
    int16s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    int16s(i).read
    int16s(i).Enabled = true
next

```

```

for i as integer = 0 to UBound(int32s)
    int32s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    int32s(i).read
    int32s(i).Enabled = true
next
for i as integer = 0 to UBound(int8s)
    int8s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    int8s(i).read
    int8s(i).Enabled = true
next
for i as integer = 0 to UBound(loneIDs)
    loneIDs(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    loneIDs(i).read
    loneIDs(i).Enabled = true
next
for i as integer = 0 to UBound(reflexives)
    reflexives(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    reflexives(i).read
    reflexives(i).Enabled = true
next
for i as integer = 0 to UBound(string128s)
    string128s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    string128s(i).read
    string128s(i).Enabled = true
next
for i as integer = 0 to UBound(string32s)
    string32s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    string32s(i).read
    string32s(i).Enabled = true
next
for i as integer = 0 to UBound(string4s)
    string4s(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    string4s(i).read
    string4s(i).Enabled = true
next
for i as integer = 0 to UBound(indexs)
    indexs(i).reflexive_offset = chunk_offset + chunk_number*chunk_size
    indexs(i).read
    indexs(i).Enabled = true
next
End Sub
bitmask16s() As bitmask16_editor_control_RWS

bitmask32s() As bitmask32_editor_control_RWS

bitmask8s() As bitmask8_editor_control_RWS

change_ok As boolean

chunk_count As Integer

chunk_number As Integer

chunk_offset As Integer

```

chunk\_size As Integer

colorARGBs() As coloraRGB\_editor\_control\_RWS

colorbytes() As colorbyte\_editor\_control\_RWS

colorRGBs() As colorRGB\_editor\_control\_RWS

dependencys() As dependency\_editor\_control\_RWS

doubles() As double\_editor\_control\_RWS

floats() As float\_editor\_control\_RWS

fs As folderItem

grand\_parent As main\_control\_RWS

id16s() As id16\_editor\_control\_RWS

id32s() As id32\_editor\_control\_RWS

id8s() As id8\_editor\_control\_RWS

indexs() As index\_editor\_control\_RWS

int16s() As int16\_editor\_control\_RWS

int32s() As int32\_editor\_control\_RWS

int8s() As int8\_editor\_control\_RWS

lonelDs() As loneID\_editor\_control\_RWS

magic As Integer

name\_offset As Integer

offset As Integer

reflexives() As reflexive\_control\_RWS

reflexive\_offset As Integer

string128s() As string128\_editor\_control\_RWS

string32s() As string32\_editor\_control\_RWS

string4s() As string4\_editor\_control\_RWS

## **reflexive\_control\_RWS Control chunk\_menu:**

Sub Change()

```

    if not change_ok then Return
    if me.ListIndex = -1 then Return
    if me.ListIndex >= chunk_count then Return
    chunk_number = me.ListIndex
    update_reflexive_offset
End Sub
End Class

```

## **Class main\_control\_RWS\_old**

Inherits ContainerControl

### **main\_control\_RWS\_old.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_magic as integer)
    fs = f
    offset = in_offset
    magic = in_magic //for passing along to reflexive editor controls
End Sub

```

### **main\_control\_RWS\_old.Destructor:**

```

Sub Destructor()
    while UBound(bitmask16s) > -1
        bitmask16s(0).close
        bitmask16s.Remove(0)
    wend
    while UBound(bitmask32s) > -1
        bitmask32s(0).close
        bitmask32s.Remove(0)
    wend
    while UBound(bitmask16s) > -1
        bitmask8s(0).close
        bitmask8s.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorARGBs(0).Close
        colorARGBs.Remove(0)
    wend
    while UBound(colorbytes) > -1
        colorbytes(0).Close
        colorbytes.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorRGBs(0).Close
        colorRGBs.Remove(0)
    wend
    while UBound(dependencys) > -1
        dependencys(0).close
        dependencys.Remove(0)
    wend
    while UBound(doubles) > -1
        doubles(0).close
        doubles.Remove(0)
    wend

```

```

while UBound(floats) > -1
    floats(0).close
    floats.Remove(0)
wend
while UBound(id16s) > -1
    id16s(0).close
    id16s.Remove(0)
wend
while UBound(id32s) > -1
    id32s(0).close
    id32s.Remove(0)
wend
while UBound(id8s) > -1
    id8s(0).close
    id8s.Remove(0)
wend
while UBound(indexs) > -1
    indexs(0).Close
    indexs.Remove(0)
wend
while UBound(int16s) > -1
    int16s(0).close
    int16s.Remove(0)
wend
while UBound(int32s) > -1
    int32s(0).close
    int32s.Remove(0)
wend
while UBound(int8s) > -1
    int8s(0).close
    int8s.Remove(0)
wend
while UBound(loneIDs) > -1
    loneIDs(0).Close
    loneIDs.Remove(0)
wend
while UBound(reflexives) > -1
    reflexives(0).close
    reflexives.Remove(0)
wend
while UBound(string128s) > -1
    string128s(0).close
    string128s.Remove(0)
wend
while UBound(string32s) > -1
    string32s(0).close
    string32s.Remove(0)
wend
while UBound(string4s) > -1
    string4s(0).close
    String4s.Remove(0)
wend
End Sub

```

main\_control\_RWS\_old.init:

Sub init(f\_xml as folderItem, in\_entity\_style as boolean, in\_show\_hidden as boolean, in\_edit\_by\_offset as boolean, byref in\_map as H1.map)

entity\_style = in\_entity\_style

show\_hidden = in\_show\_hidden

edit\_by\_offset = in\_edit\_by\_offset

map = in\_map

load(f\_xml)

End Sub

### **main\_control\_RWS\_old.load:**

Sub load(f\_xml as folderItem)

while UBound(bitmask16s) > -1

bitmask16s(0).close

bitmask16s.Remove(0)

wend

while UBound(bitmask32s) > -1

bitmask32s(0).close

bitmask32s.Remove(0)

wend

while UBound(bitmask16s) > -1

bitmask8s(0).close

bitmask8s.Remove(0)

wend

while UBound(colorARGBs) > -1

colorARGBs(0).Close

colorARGBs.Remove(0)

wend

while UBound(colorbytes) > -1

colorbytes(0).Close

colorbytes.Remove(0)

wend

while UBound(colorARGBs) > -1

colorRGBs(0).Close

colorRGBs.Remove(0)

wend

while UBound(dependencys) > -1

dependencys(0).close

dependencys.Remove(0)

wend

while UBound(doubles) > -1

doubles(0).close

doubles.Remove(0)

wend

while UBound(floats) > -1

floats(0).close

floats.Remove(0)

wend

while UBound(id16s) > -1

id16s(0).close

id16s.Remove(0)

wend

while UBound(id32s) > -1



```

    id32s(0).close
    id32s.Remove(0)
wend
while UBound(id8s) > -1
    id8s(0).close
    id8s.Remove(0)
wend
while UBound(indexs) > -1
    indexs(0).Close
    indexs.Remove(0)
wend
while UBound(int16s) > -1
    int16s(0).close
    int16s.Remove(0)
wend
while UBound(int32s) > -1
    int32s(0).close
    int32s.Remove(0)
wend
while UBound(int8s) > -1
    int8s(0).close
    int8s.Remove(0)
wend
while UBound(loneIDs) > -1
    loneIDs(0).Close
    loneIDs.Remove(0)
wend
while UBound(reflexives) > -1
    reflexives(0).close
    reflexives.Remove(0)
wend
while UBound(string128s) > -1
    string128s(0).close
    string128s.Remove(0)
wend
while UBound(string32s) > -1
    string32s(0).close
    string32s.Remove(0)
wend
while UBound(string4s) > -1
    string4s(0).close
    String4s.Remove(0)
wend

redim bitmask16s(-1)
redim bitmask32s(-1)
redim bitmask8s(-1)
redim colorARGBs(-1)
redim colorbytes(-1)
redim colorRGBs(-1)
redim dependencys(-1)
redim doubles(-1)
redim floats(-1)
redim id16s(-1)

```

```

redim id32s(-1)
redim id8s(-1)
redim indexs(-1)
redim int16s(-1)
redim int32s(-1)
redim int8s(-1)
redim lonelDs(-1)
redim reflexives(-1)
redim string128s(-1)
redim string32s(-1)
redim string4s(-1)

change_ok = false
dim f_top as FolderItem = GetFolderItem("").Child("Plugins")
dim class_str as string = f_xml.name.mid(1,4)
PopupMenu1.DeleteAllRows
for i as integer = 1 to f_top.Count
    dim f_xml_folder as FolderItem = f_top.item(i)
    if f_xml_folder.exists and f_xml_folder.Directory then
        if f_xml_folder.Child(class_str + ".ent").Exists then
            dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".ent")
            PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
            PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
        end
        if f_xml_folder.Child(class_str + ".xml").Exists then
            dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".xml")
            PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
            PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
        end
    end
end
next
for i as integer = 0 to PopupMenu1.ListCount - 1
    if PopupMenu1.RowTag(i) = f_xml.Parent.Name + ":" + f_xml.Name then
        PopupMenu1.ListIndex = i
        exit for i
    end
end
next
change_ok = true

dim plugin_info as new PluginLibUniversal
dim ref(-1) as integer
if entity_style then
    plugin_info.refresh_as_Ent(f_xml, false, ref, show_hidden)
else
    plugin_info.refresh_as_HMT(f_xml, false, ref)
end

//so organize the various editing components of the main reflexive
dim index_list(-1) as string
dim offset_list(-1) as integer
dim order_list(-1) as integer
for i as integer = 0 to UBound(plugin_info.bitmask16_offset)
    index_list.Append("bitmask16:" + str(i))
    offset_list.Append plugin_info.bitmask16_offset(i)

```

```

    order_list.Append plugin_info.bitmask16_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask32_offset)
    index_list.Append("bitmask32:" + str(i))
    offset_list.Append plugin_info.bitmask32_offset(i)
    order_list.Append plugin_info.bitmask32_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask8_offset)
    index_list.Append("bitmask8:" + str(i))
    offset_list.Append plugin_info.bitmask8_offset(i)
    order_list.Append plugin_info.bitmask8_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorARGB_offset)
    index_list.Append("colorARGB:" + str(i))
    offset_list.Append plugin_info.colorARGB_offset(i)
    order_list.Append plugin_info.colorARGB_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorbyte_offset)
    index_list.Append("colorbyte:" + str(i))
    offset_list.Append plugin_info.colorbyte_offset(i)
    order_list.Append plugin_info.colorbyte_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorRGB_offset)
    index_list.Append("colorRGB:" + str(i))
    offset_list.Append plugin_info.colorRGB_offset(i)
    order_list.Append plugin_info.colorRGB_order(i)
next
for i as integer = 0 to UBound(plugin_info.dependency_offset)
    index_list.Append("dependency:" + str(i))
    offset_list.Append plugin_info.dependency_offset(i)
    order_list.Append plugin_info.dependency_order(i)
next
for i as integer = 0 to UBound(plugin_info.double_offset)
    index_list.Append("double:" + str(i))
    offset_list.Append plugin_info.double_offset(i)
    order_list.Append plugin_info.double_order(i)
next
for i as integer = 0 to UBound(plugin_info.float_offset)
    index_list.Append("float:" + str(i))
    offset_list.Append plugin_info.float_offset(i)
    order_list.Append plugin_info.float_order(i)
next
for i as integer = 0 to UBound(plugin_info.id16_offset)
    index_list.Append("id16:" + str(i))
    offset_list.Append plugin_info.id16_offset(i)
    order_list.Append plugin_info.id16_order(i)
next
for i as integer = 0 to UBound(plugin_info.id32_offset)
    index_list.Append("id32:" + str(i))
    offset_list.Append plugin_info.id32_offset(i)
    order_list.Append plugin_info.id32_order(i)
next
for i as integer = 0 to UBound(plugin_info.id8_offset)
    index_list.Append("id8:" + str(i))

```

```

    offset_list.Append plugin_info.id8_offset(i)
    order_list.Append plugin_info.id8_order(i)
next
for i as integer = 0 to UBound(plugin_info.index_offset)
    index_list.Append("index:" + str(i))
    offset_list.Append plugin_info.index_offset(i)
    order_list.Append plugin_info.index_order(i)
next
for i as integer = 0 to UBound(plugin_info.int16_offset)
    index_list.Append("int16:" + str(i))
    offset_list.Append plugin_info.int16_offset(i)
    order_list.Append plugin_info.int16_order(i)
next
for i as integer = 0 to UBound(plugin_info.int32_offset)
    index_list.Append("int32:" + str(i))
    offset_list.Append plugin_info.int32_offset(i)
    order_list.Append plugin_info.int32_order(i)
next
for i as integer = 0 to UBound(plugin_info.int8_offset)
    index_list.Append("int8:" + str(i))
    offset_list.Append plugin_info.int8_offset(i)
    order_list.Append plugin_info.int8_order(i)
next
for i as integer = 0 to UBound(plugin_info.loneID_offset)
    index_list.Append("loneID:" + str(i))
    offset_list.Append plugin_info.loneID_offset(i)
    order_list.Append plugin_info.loneID_order(i)
next
for i as Integer = 0 to UBound(plugin_info.reflexive_offset)
    index_list.Append("reflexive:" + str(i))
    offset_list.Append plugin_info.reflexive_offset(i)
    order_list.Append plugin_info.reflexive_order(i)
next
for i as integer = 0 to UBound(plugin_info.string128_offset)
    index_list.Append("string128:" + str(i))
    offset_list.Append plugin_info.string128_offset(i)
    order_list.Append plugin_info.string128_order(i)
next
for i as integer = 0 to UBound(plugin_info.string32_offset)
    index_list.Append("string32:" + str(i))
    offset_list.Append plugin_info.string32_offset(i)
    order_list.Append plugin_info.string32_order(i)
next
for i as integer = 0 to UBound(plugin_info.string4_offset)
    index_list.Append("string4:" + str(i))
    offset_list.Append plugin_info.string4_offset(i)
    order_list.Append plugin_info.string4_order(i)
next

if edit_by_offset then
    offset_list.sortwith(index_list)
else
    order_list.sortwith(index_list, offset_list)
end

```

```

//now create the individual controls and add them to the main window control
dim temp_height as integer = 14
for i as integer = 0 to UBound(offset_list)
    dim temp_split_str() as string = split(index_list(i), ":")
    select case temp_split_str(0)

case "bitmask16"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask16_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask16_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_bitmask as new bitmask16_editor_control_RWS(fs, offset_list(i),
    plugin_info.bitmask16_data(val(temp_split_str(1))).labels, temp_ints)
    temp_bitmask.name_text.Text = plugin_info.bitmask16_name(val(temp_split_str(1)))
    temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
    bitmask16s.Append temp_bitmask
    temp_height = temp_height + 12 + temp_bitmask.Height

case "bitmask32"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask32_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask32_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_bitmask as new bitmask32_editor_control_RWS(fs, offset_list(i),
    plugin_info.bitmask32_data(val(temp_split_str(1))).labels, temp_ints)
    temp_bitmask.name_text.Text = plugin_info.bitmask32_name(val(temp_split_str(1)))
    temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
    bitmask32s.Append temp_bitmask
    temp_height = temp_height + 12 + temp_bitmask.Height

case "bitmask8"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask8_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask8_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_bitmask as new bitmask8_editor_control_RWS(fs, offset_list(i),
    plugin_info.bitmask8_data(val(temp_split_str(1))).labels, temp_ints)
    temp_bitmask.name_text.Text = plugin_info.bitmask8_name(val(temp_split_str(1)))
    temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
    bitmask8s.Append temp_bitmask
    temp_height = temp_height + 12 + temp_bitmask.Height

case "colorARGB"
    dim temp_color as new colorARGB_editor_control_RWS(fs, offset_list(i))
    temp_color.name_text.Text = plugin_info.colorARGB_name(val(temp_split_str(1)))
    temp_color.EmbedWithin(Canvas1, 20, temp_height)
    colorARGBs.Append temp_color
    temp_height = temp_height + 12 + temp_color.Height

case "colorbyte"
    dim temp_color as new colorbyte_editor_control_RWS(fs, offset_list(i))
    temp_color.name_text.Text = plugin_info.colorbyte_name(val(temp_split_str(1)))
    temp_color.EmbedWithin(Canvas1, 20, temp_height)

```

```

colorbytes.Append temp_color
temp_height = temp_height + 12 + temp_color.Height

case "colorRGB"
    dim temp_color as new colorRGB_editor_control_RWS(fs, offset_list(i))
    temp_color.name_text.Text = plugin_info.colorRGB_name(val(temp_split_str(1)))
    temp_color.EmbedWithin(Canvas1, 20, temp_height)
    colorRGBs.Append temp_color
    temp_height = temp_height + 12 + temp_color.Height

case "dependency"
    dim temp_dep as new dependency_editor_control_RWS(fs, offset_list(i), map)
    temp_dep.name_text.Text = plugin_info.dependency_name(val(temp_split_str(1)))
    temp_dep.EmbedWithin(Canvas1, 20, temp_height)
    dependencycs.Append temp_dep
    temp_height = temp_height + 12 + temp_dep.Height

case "double"
    dim temp_double as new double_editor_control_RWS(fs, offset_list(i))
    temp_double.name_text.Text = plugin_info.double_name(val(temp_split_str(1)))
    temp_double.EmbedWithin(Canvas1, 20, temp_height)
    doubles.Append temp_double
    temp_height = temp_height + 12 + temp_double.Height

case "float"
    dim temp_float as new float_editor_control_RWS(fs, offset_list(i))
    temp_float.name_text.Text = plugin_info.float_name(val(temp_split_str(1)))
    temp_float.EmbedWithin(Canvas1, 20, temp_height)
    floats.Append temp_float
    temp_height = temp_height + 12 + temp_float.Height

case "id16"
    dim temp_ints(-1) as Int16
    for j as integer = 0 to UBound(plugin_info.id16_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id16_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id16_editor_control_RWS(fs, offset_list(i),
    plugin_info.id16_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id16_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id16s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

case "id32"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.id32_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id32_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id32_editor_control_RWS(fs, offset_list(i),
    plugin_info.id32_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id32_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id32s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

```

```

case "id8"
    dim temp_ints(-1) as Int8
    for j as integer = 0 to UBound(plugin_info.id8_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id8_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id8_editor_control_RWS(fs, offset_list(i),
    plugin_info.id8_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id8_name(val(temp_split_str(1)))
    temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id8s.Append temp_enum
    temp_height = temp_height + 12 + temp_enum.Height

case "index"
    dim temp_index as new index_editor_control_RWS(fs, offset_list(i),
    plugin_info.index_data(val(temp_split_str(1))))
    temp_index.name_text.Text = plugin_info.index_name(val(temp_split_str(1)))
    temp_index.EmbedWithin(Canvas1, 20, temp_height)
    indexes.Append temp_index
    temp_height = temp_height + 12 + temp_index.Height

case "int16"
    dim temp_int as new int16_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int16_name(val(temp_split_str(1)))
    temp_int.EmbedWithin(Canvas1, 20, temp_height)
    int16s.Append temp_int
    temp_height = temp_height + 12 + temp_int.Height

case "int32"
    dim temp_int as new int32_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int32_name(val(temp_split_str(1)))
    temp_int.EmbedWithin(Canvas1, 20, temp_height)
    int32s.Append temp_int
    temp_height = temp_height + 12 + temp_int.Height

case "int8"
    dim temp_int as new int8_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int8_name(val(temp_split_str(1)))
    temp_int.EmbedWithin(Canvas1, 20, temp_height)
    int8s.Append temp_int
    temp_height = temp_height + 12 + temp_int.Height

case "lonelD"
    dim temp_dep as new lonelD_editor_control_RWS(fs, offset_list(i), map)
    temp_dep.name_text.Text = plugin_info.lonelD_name(val(temp_split_str(1)))
    temp_dep.EmbedWithin(Canvas1, 20, temp_height)
    LonelDs.Append temp_dep
    temp_height = temp_height + 12 + temp_dep.Height

case "reflexive"
    dim temp_reflexive as new reflexive_control_RWS(fs, offset_list(i),
    plugin_info.reflexives(val(temp_split_str(1))).this_reflexive_size, magic, _
    plugin_info.reflexive_name_offset(val(temp_split_str(1))), offset)
    temp_reflexive.groupbox1.caption = plugin_info.reflexive_name(val(temp_split_str(1)))

```

```

temp_reflexive.EmbedWithin(Canvas1, 20, temp_Height)
break
//fix this
//temp_reflexive.add_granddaddy(me)
temp_reflexive.init(plugin_info.reflexives(val(temp_split_str(1))), edit_by_offset, map)
reflexives.Append temp_reflexive
temp_height = temp_height + 12 + temp_reflexive.Height

case "string128"
    dim temp_string as new string128_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string128_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(Canvas1, 20, temp_height)
    string128s.Append temp_string
    temp_height = temp_height + 12 + temp_string.Height

case "string32"
    dim temp_string as new string32_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string32_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(Canvas1, 20, temp_height)
    string32s.Append temp_string
    temp_height = temp_height + 12 + temp_string.Height

case "string 4"
    dim temp_string as new string4_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string4_name(val(temp_split_str(1)))
    temp_string.EmbedWithin(Canvas1, 20, temp_height)
    string4s.Append temp_string
    temp_height = temp_height + 12 + temp_string.Height

end select
next

//so now all of the stuff has been added to the control
temp_height = temp_height + 8 // -12 from the last control +20 space it from the bottom
me.Height = temp_height + 49
dim temp_width as integer = 400 //the default width of editing controls
for i as integer = 0 to UBound(reflexives)
    if reflexives(i).Width > temp_width then
        temp_width = reflexives(i).Width
    end
next
temp_width = temp_width + 10
Canvas1.Refresh
Canvas1.Width = temp_width
me.Width = temp_width

read
//update_indexs
End Sub

main_control_RWS_old.read:
Sub read()
    update_reflexive_offset
    update_indexs

```



```
    update_reflexive_offset
End Sub
```

### **main\_control\_RWS\_old.update\_indexs:**

```
Sub update_indexs()
    //this only works for top level indexs
    for i as integer = 0 to UBound(indexs)
        dim temp_path() as string = split(indexs(i).path, ".")
        dim temp_reflexives() as reflexive_control_RWS = reflexives
        for j as integer = 1 to UBound(temp_path) - 1
            dim ref_name as string = temp_path(j)
            for k as integer = 0 to UBound(temp_reflexives)
                if temp_reflexives(k).groupbox1.caption = ref_name then
                    temp_reflexives = temp_reflexives(k).reflexives
                    exit for k
                end
            next
        next
        for j as integer = 0 to UBound(temp_reflexives)
            if temp_reflexives(j).groupbox1.caption = temp_path(UBound(temp_path)) then
                indexs(i).reflexive_pointed_at = temp_reflexives(j)
                exit for j
            end
        next
    next

    for i as integer = 0 to UBound(reflexives)
        reflexives(i).update_indexs
    next
End Sub
```

### **main\_control\_RWS\_old.update\_reflexive\_offset:**

```
Sub update_reflexive_offset()
    //their own "offset" is the local offset, the reflexive_offset is where the reflexive starts
    for i as integer = 0 to UBound(bitmask16s)
        bitmask16s(i).reflexive_offset = offset
        bitmask16s(i).read
        bitmask16s(i).Enabled = true
    next
    for i as integer = 0 to UBound(bitmask32s)
        bitmask32s(i).reflexive_offset = offset
        bitmask32s(i).read
        bitmask32s(i).Enabled = true
    next
    for i as integer = 0 to UBound(bitmask8s)
        bitmask8s(i).reflexive_offset = offset
        bitmask8s(i).read
        bitmask8s(i).Enabled = true
    next
    for i as integer = 0 to UBound(colorARGBs)
        colorARGBs(i).reflexive_offset = offset
        colorARGBs(i).read
        colorARGBs(i).Enabled = true
    next
End Sub
```

```

for i as integer = 0 to UBound(colorbytes)
    colorbytes(i).reflexive_offset = offset
    colorbytes(i).read
    colorbytes(i).Enabled = true
next
for i as integer = 0 to UBound(colorRGBs)
    colorRGBs(i).reflexive_offset = offset
    colorRGBs(i).read
    colorRGBs(i).Enabled = true
next
for i as integer = 0 to UBound(dependencys)
    dependencys(i).reflexive_offset = offset
    dependencys(i).read
    dependencys(i).Enabled = true
next
for i as integer = 0 to UBound(doubles)
    doubles(i).reflexive_offset = offset
    doubles(i).read
    doubles(i).Enabled = true
next
for i as integer = 0 to UBound(floats)
    floats(i).reflexive_offset = offset
    floats(i).read
    floats(i).Enabled = true
next
for i as integer = 0 to UBound(id16s)
    id16s(i).reflexive_offset = offset
    id16s(i).read
    id16s(i).Enabled = true
next
for i as integer = 0 to UBound(id32s)
    id32s(i).reflexive_offset = offset
    id32s(i).read
    id32s(i).Enabled = true
next
for i as integer = 0 to UBound(id8s)
    id8s(i).reflexive_offset = offset
    id8s(i).read
    id8s(i).Enabled = true
next
//would normally put indexes here but they need to be after reflexives
for i as integer = 0 to UBound(int16s)
    int16s(i).reflexive_offset = offset
    int16s(i).read
    int16s(i).Enabled = true
next
for i as integer = 0 to UBound(int32s)
    int32s(i).reflexive_offset = offset
    int32s(i).read
    int32s(i).Enabled = true
next
for i as integer = 0 to UBound(int8s)
    int8s(i).reflexive_offset = offset
    int8s(i).read

```

```

    int8s(i).Enabled = true
next
for i as integer = 0 to UBound(LoneIDs)
    loneIDs(i).reflexive_offset = offset
    loneIDs(i).read
    loneIDs(i).Enabled = true
next
for i as integer = 0 to UBound(reflexives)
    reflexives(i).reflexive_offset = offset
    reflexives(i).read
    reflexives(i).Enabled = true
next
for i as integer = 0 to UBound(string128s)
    string128s(i).reflexive_offset = offset
    string128s(i).read
    string128s(i).Enabled = true
next
for i as integer = 0 to UBound(string32s)
    string32s(i).reflexive_offset = offset
    string32s(i).read
    string32s(i).Enabled = true
next
for i as integer = 0 to UBound(string4s)
    string4s(i).reflexive_offset = offset
    string4s(i).read
    string4s(i).Enabled = true
next
for i as integer = 0 to UBound(indexs)
    indexs(i).reflexive_offset = offset
    indexs(i).read
    indexs(i).Enabled = true
next
End Sub
bitmask16s() As bitmask16_editor_control_RWS

bitmask32s() As bitmask32_editor_control_RWS

bitmask8s() As bitmask8_editor_control_RWS

change_ok As boolean

colorARGBs() As colorARGB_editor_control_RWS

colorbytes() As colorbyte_editor_control_RWS

colorRGBs() As colorRGB_editor_control_RWS

dependencys() As dependency_editor_control_RWS

doubles() As double_editor_control_RWS

edit_by_offset As boolean

entity_style As boolean

```

floats() As float\_editor\_control\_RWS

fs As FolderItem

id16s() As id16\_editor\_control\_RWS

id32s() As id32\_editor\_control\_RWS

id8s() As id8\_editor\_control\_RWS

indexs() As index\_editor\_control\_RWS

index\_data As string

int16s() As int16\_editor\_control\_RWS

int32s() As int32\_editor\_control\_RWS

int8s() As int8\_editor\_control\_RWS

lonelDs() As lonelD\_editor\_control\_RWS

magic As Integer

map As h1.map

offset As Integer

reflexives() As reflexive\_control\_RWS

show\_hidden As boolean

string128s() As string128\_editor\_control\_RWS

string32s() As string32\_editor\_control\_RWS

string4s() As string4\_editor\_control\_RWS

## **main\_control\_RWS\_old Control PopupMenu1:**

Sub Change()

if not change\_ok then return

dim temp\_strs() as string = split(me.RowTag(me.ListIndex), ":")

if temp\_strs.ubound <> 1 then return

dim folder as FolderItem = GetFolderItem("").Child("Plugins").Child(temp\_strs(0))

if Folder.Exists and Folder.Directory then

dim plugin as FolderItem = Folder.Child(temp\_strs(1))

if plugin.Exists then

dim temp\_split() as string = plugin.name.split(".")

dim temp\_str as string

if temp\_split.Ubound < 1 then

'dim temp\_split2() as string = plugin.AbsolutePath.Split(":")

'dim temp\_str2 as string = temp\_split2(temp\_split2.Ubound)

```

'dim temp_split3() as string = temp_str2.split(".")
'if temp_split3.Ubound < 1 then
'break
'errorbox("There was an error determining plugin extension type")
'return
'end
'dim temp_str3 as string = temp_split3(1)
'#if buildtargetwindows
'temp_str3 = temp_str3.left(temp_str3.length - 1)
'#endif
'temp_str = temp_str3
break
return
else
temp_str = temp_split(1)
end
select case temp_str.Uppercase
case "ENT"
entity_style = true
case "XML"
entity_style = false
end select
load(plugin)
end
end
End Sub
End Class

```

## **Class no\_plugin\_control**

Inherits ContainerControl

End Class

## **Class bitm\_editor\_control\_RW**

Inherits ContainerControl

### **bitm\_editor\_control\_RW.init:**

```

Sub init(in_bitmap as bitmap_class_RW, in_name as string)
bitm_data = in_bitmap
bitm_data.read

name = in_name
change_ok = false
image_popup.DeleteAllRows
cube_popup.DeleteAllRows
mip_popup.DeleteAllRows
cube_text.Enabled = false
mip_text.Enabled = false
for i as integer = 0 to UBound(bitm_data.image_info)
image_popup.AddRow(str(i+1) + ": " + str(bitm_data.image_info(i).width) _
+ "x" + str(bitm_data.image_info(i).height) )
next

```

```
    change_ok = true
    image_popup.ListIndex = 0
End Sub
bitm As bitm_image
```

```
bitm_data As bitmap_class_RW
```

```
change_ok As boolean
```

```
last_loaded As Integer
```

```
main_full As picture
```

```
mask_full As picture
```

```
name As string
```

### **bitm\_editor\_control\_RW Control image\_popup:**

```
Sub Change()
    if not change_ok then Return
    if me.ListIndex = -1 then Return

    //file info
    dim index as integer = me.ListIndex
    if BitAnd(bitm_data.image_info(index).flags, &h100) = &h100 then
        //external
        source_text.Text = "Source: Bitmaps.map"
        dds_eof_check.Enabled = true
    else
        //internal
        source_text.Text = "Source: Internal Bitmaps"
        dds_eof_check.Enabled = false
        dds_eof_check.Value = false
    end
    select case bitm_data.image_info(index).format
    case &h0 //A8
        format_text.Text = "Format: A8"

    case &h1 //Y8
        format_text.Text = "Format: Y8"

    case &h2 //AY8
        format_text.Text = "Format: AY8"

    case &h3 //A8Y8
        format_text.Text = "Format: A8Y8"

    case &h6 //R5G6B5
        format_text.Text = "Format: R5G6B5"

    case &h8 //A1R5G5B5
        format_text.Text = "Format: A1R5G5B5"
```

```

case &h9 //A4R4G4B4
    format_text.Text = "Format: A4R4G4B4"

case &hA //X8R8G8B8
    format_text.Text = "Format: X8R8G8B8"

case &hB //A8R8G8B8
    format_text.Text = "Format: A8R8G8B8"

case &hE //DXT1
    format_text.Text = "Format: DXT1"

case &hF //DXT2and3
    format_text.Text = "Format: DXT2/3"

case &h10 //DXT4and5
    format_text.Text = "Format: DXT4/5"

case &h11 //P8
    format_text.Text = "Format: P8"

end select

//deal with cube_map-ness
if bitm_data.image_info(index).type = 2 then
    //cube map
    cube_popup.Enabled = true
    cube_text.Enabled = true
    change_ok = false
    cube_popup.DeleteAllRows
    cube_popup.AddRow("Postive X")
    cube_popup.AddRow("Negative X")
    cube_popup.AddRow("Postive Y")
    cube_popup.AddRow("Negative Y")
    cube_popup.AddRow("Postive Z")
    cube_popup.AddRow("Negative Z")
    change_ok = true
    cube_popup.ListIndex = 0
else
    //2D most likely
    change_ok = false
    cube_popup.DeleteAllRows
    change_ok = true
    cube_popup.Enabled = false
    cube_text.Enabled = false
end

//for now
import_push.Enabled = false
select case bitm_data.image_info(index).type
case 0
    //2D
    import_push.Enabled = true
case 1

```

```

import_push.Enabled = true
case 2
import_push.Enabled = true
end select

//delete all layer data for now let it be updated by the mipmap control
change_ok = false
layer_text.Enabled = false
layer_popup.Enabled = false
layer_popup.DeleteAllRows
change_ok = true

//deal with mip_map-ness
if bitm_data.image_info(index).num_mipmaps > 0 then
mip_popup.Enabled = true
mip_text.Enabled = true
change_ok = false
mip_popup.DeleteAllRows
mip_popup.AddRow("1: Main")
for i as integer = 1 to bitm_data.image_info(index).num_mipmaps
mip_popup.AddRow(str(i+1)+": " + str(Ceil(bitm_data.image_info(index).width/(2^i))) _
+ "x" + str(Ceil(bitm_data.image_info(index).height/(2^i))) )
next
change_ok = true
mip_popup.ListIndex = 0
else
change_ok = false
mip_popup.DeleteAllRows
change_ok = true
mip_popup.Enabled = false
mip_text.Enabled = false
end
End Sub

```

### **bitm\_editor\_control\_RW Control mip\_popup:**

```

Sub Change()
//going to need to add some stuff for layer ness

dim image_index as integer = image_popup.ListIndex
dim mip_index as integer = me.ListIndex

//volume texture
if bitm_data.image_info(image_index).type = 1 then
change_ok = false
layer_text.Enabled = true
layer_popup.Enabled = true
layer_popup.DeleteAllRows

for i as integer = 1 to max(bitm_data.image_info(image_index).depth/(2^mip_index), 1)
layer_popup.AddRow("Layer: " + str(i))
next
layer_popup.ListIndex = 0
change_ok = true
end

```



End Sub

### **bitm\_editor\_control\_RW Control main\_well:**

Function MouseDown(X As Integer, Y As Integer) As Boolean

```
if me.Image = nil then Return true
if main_full = nil then Return true
dim w as new bitm_preview_window
w.Height = main_full.Height
w.Width = main_full.Width
w.Backdrop = main_full
w.show
return true
```

End Function

### **bitm\_editor\_control\_RW Control mask\_well:**

Function MouseDown(X As Integer, Y As Integer) As Boolean

```
if me.Image = nil then Return true
if mask_full = nil then Return true
dim w as new bitm_preview_window
w.Height = mask_full.Height
w.Width = mask_full.Width
w.Backdrop = mask_full
w.show
return true
```

End Function

### **bitm\_editor\_control\_RW Control dds\_extract\_push:**

Sub Action()

```
dim image_index as integer = image_popup.ListIndex
```

```
Dim ddsType as New FileType
```

```
ddsType.Name = "image/dds"
```

```
//3f3f3f3f
```

```
ddsType.MacType = chr(&h3f) + chr(&h3f) + chr(&h3f) + chr(&h3f)
```

```
ddsType.Extensions = "dds"
```

```
dim f as FolderItem = GetSaveFolderItem(ddsType, name + "." + str(1+image_index) + ".dds")
```

```
if f = nil then Return
```

```
dim output as MemoryBlock = bitm_data.extract_DDS(image_index)
```

```
dim bw as BinaryStream = f.CreateBinaryFile(ddsType)
```

```
bw.LittleEndian = true
```

```
bw.Write(output)
```

```
bw.Close
```

```
f.ExtensionVisible = true
```

End Sub

### **bitm\_editor\_control\_RW Control dds\_inject\_push:**

Sub Action()

```
dim image_index as integer = image_popup.ListIndex
```

```
Dim ddsType as New FileType
```

```
ddsType.Name = "image/dds"
```

```
//3f3f3f3f
```

```
ddsType.MacType = chr(&h3f) + chr(&h3f) + chr(&h3f) + chr(&h3f)
```

```

ddsType.Extensions = "dds"

dim f as FolderItem = GetOpenFolderItem(ddsType)
if f = nil then Return

dim data_block as new MemoryBlock(0)
dim bw as new BinaryStream(data_block)
bw.littleendian = true
bw.position = 0
dim br as BinaryStream = f.openasbinaryfile
br.littleendian = true
br.position = 0
dim temp_len as integer = br.length
for i as integer = 1 to temp_len
    bw.writeint8(br.readint8)
    if data_block.size > temp_len then
        //break
    end
next
data_block.size = br.length
dim output as integer = bitm_data.inject_DDS(image_index, data_block, dds_eof_check.value, name)

select case output
case 0
    MsgBox("Image Injected Successfully!")
case -1
    errorbox("Error: Unable to inject image")
case -2
    errorbox("Error: File too large to inject")
case -3
    errorbox("Error: File is not a valid DDS file")
case -4
    errorbox("Error: Image dimensions too large")
case -5
    dim output_str as string = "Error: Wrong format. Should be "
    select case bitm_data.image_info(image_index).format
    case int32(BitmFormat.A1R5G5B5)
        output_str = output_str + "A1R5G5B5"
    case int32(BitmFormat.A4R4G4B4)
        output_str = output_str + "A4R4G4B4"
    case int32(BitmFormat.A8)
        output_str = output_str + "A8"
    case int32(BitmFormat.A8R8G8B8)
        output_str = output_str + "A8R8G8B8"
    case int32(BitmFormat.A8Y8)
        output_str = output_str + "A8Y8"
    case int32(BitmFormat.AY8)
        output_str = output_str + "AY8"
    case int32(BitmFormat.DXT1)
        output_str = output_str + "DXT1"
    case int32(BitmFormat.DXT2_3)
        output_str = output_str + "DXT2/3"
    case int32(BitmFormat.DXT4_5)
        output_str = output_str + "DXT4/5"
    end select
end select

```

```

    case int32(BitmFormat.P8)
        output_str = output_str + "P8"
    case int32(BitmFormat.R5G6B5)
        output_str = output_str + "R5G6B5"
    case int32(BitmFormat.X8R8G8B8)
        output_str = output_str + "X8R8G8B8"
    case int32(BitmFormat.Y8)
        output_str = output_str + "Y8"
    end select
    errorbox(output_str)
case -6
    errorbox("Error: Image does not contain cubemap data")
case -7
    errorbox("Error: Image does not contain volume texture data")
end select

//make it reload the data based on any changes
image_popup.ListIndex = -1
image_popup.ListIndex = image_index

//0: Succeeded
//-1: Failed, no known reason
//-2: Failed, file size too large
//-3: Failed, bad header
//-4: Failed, image dimensions too large
//-5: Failed, image wrong format
//-6: Failed, needs to be a cubemap
//-7: Failed, needs to be a volume texture
End Sub
bitm_editor_control_RW Control export_push:

Sub Action()
    dim image_index as integer = image_popup.ListIndex

    //read the single image
    bitm_data.read_bitmap(image_index)

    dim cont as boolean = true
    select case bitm_data.image_info(image_index).type

    case 2
        //cube map
        for i as integer = 0 to 5
            if cont then
                cont = ExportPicture(bitm_data.cube_images(image_index).face(i).image)
                if bitm_data.cube_images(0).face(i).mask_flag AND cont then
                    cont = ExportPicture(bitm_data.cube_images(0).face(i).mask)
                end
            end
        next
        if cont then
            if bitm_data.cube_images(0).face(0).mask_flag then
                MsgBox("Note: The order of images saved is as follows: " + EndOfLine + _
                    "Postive X" + EndOfLine + _

```

```

        "Postive X Mask" + EndOfLine + _
        "Negative X" + EndOfLine + _
        "Negative X Mask" + EndOfLine + _
        "Postive Y" + EndOfLine + _
        "Postive Y Mask" + EndOfLine + _
        "Negative Y" + EndOfLine + _
        "Negative Y Mask" + EndOfLine + _
        "Postive Z" + EndOfLine + _
        "Postive Z Mask" + EndOfLine + _
        "Negative Z" + EndOfLine + _
        "Negative Z Mask")
    else
        MsgBox("Note: The order of images saved is as follows: " + EndOfLine + _
        "Postive X" + EndOfLine + _
        "Negative X" + EndOfLine + _
        "Postive Y" + EndOfLine + _
        "Negative Y" + EndOfLine + _
        "Postive Z" + EndOfLine + _
        "Negative Z")
    end
end
end

case 1

cont = ExportPicture(bitm_data.images(0).image)
if cont and bitm_data.images(0).mask_flag then
    cont = ExportPicture(bitm_data.images(0).mask)
end
for i as integer = 0 to UBound(bitm_data.images(0).layers)
    cont = ExportPicture(bitm_data.images(0).layers(i).image)
    if cont and bitm_data.images(0).layers(i).mask_flag then
        cont = ExportPicture(bitm_data.images(0).layers(i).mask)
    end
next
if cont then
    if bitm_data.images(0).mask_flag then
        MsgBox("Note: The order of images saved is the layer followed by the mask for that layer")
    else
        MsgBox("Note: The order of images saved is simply the layers of the image in order")
    end
end
end

case 0
cont = ExportPicture(bitm_data.images(0).image)
if cont and bitm_data.images(0).mask_flag then
    cont = ExportPicture(bitm_data.images(0).mask)
    if cont then
        MsgBox("Note: The first image that was saved is the main image. " + _
        "The second image saved was the image mask for the main image")
    end
end
end
end
End Sub

bitm_editor_control_RW Control import_push:

```

Sub Action()

```
dim image_index as integer = image_popup.ListIndex
dim temp_height as integer = bitm_data.image_info(image_index).height
dim temp_width as integer = bitm_data.image_info(image_index).width

select case bitm_data.image_info(image_index).type
case 0
  //2D image
  select case bitm_data.image_info(image_index).format
  case &h0, &h1, &h6, &hA, &hE, &h11 //A8, Y8, R5G6B5, X8R8G8B8, DXT1, P8
    //image no mask
    dim w as new bitm_import_single_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  case &h2, &h3, &h8, &h9, &hB, &hF, &h10 //AY8, A8Y8, A1R5G5B5, A4R4G4B4, A8R8G8B8, DXT2/3, DXT4/5
    //image + mask
    dim w as new bitm_import_single_mask_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  end select

case 1
  //volume images

  select case bitm_data.image_info(image_index).format
  case &h0, &h1, &h6, &hA, &hE, &h11 //A8, Y8, R5G6B5, X8R8G8B8, DXT1, P8
    //image no mask
    dim w as new bitm_import_volume_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  case &h2, &h3, &h8, &h9, &hB, &hF, &h10 //AY8, A8Y8, A1R5G5B5, A4R4G4B4, A8R8G8B8, DXT2/3, DXT4/5
    //image + mask
    dim w as new bitm_import_volume_mask_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  end select

case 2
  //cube maps
  select case bitm_data.image_info(image_index).format
  case &h0, &h1, &h6, &hA, &hE, &h11 //A8, Y8, R5G6B5, X8R8G8B8, DXT1, P8
    //image no mask
    dim w as new bitm_import_cube_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  case &h2, &h3, &h8, &h9, &hB, &hF, &h10 //AY8, A8Y8, A1R5G5B5, A4R4G4B4, A8R8G8B8, DXT2/3, DXT4/5
    //image + mask
    dim w as new bitm_import_cube_mask_RW
    w.show
    w.init(bitm_data.image_info(image_index), bitm_data.f, bitm_data.bitmaps_f)
  end select

end select
```

End Sub

## **bitm\_editor\_control\_RW Control Load\_push:**

Sub Action()

```
dim image_index as integer = image_popup.ListIndex
```

```
//read the single image
```

```
if last_loaded <> image_index then
```

```
    bitm_data.read_bitmap(image_index)
```

```
    last_loaded = image_index
```

```
end
```

```
//clear out the current images
```

```
main_well.Image = nil
```

```
mask_well.Image = nil
```

```
main_full = nil
```

```
mask_full = nil
```

```
//take care of cubemaps
```

```
if bitm_data.image_info(image_index).type = 2 then
```

```
    //cube map
```

```
    dim cube_index as integer = cube_popup.ListIndex
```

```
    bitm = bitm_data.cube_images(0).face(cube_index) //0 because of read_bitmap
```

```
else
```

```
    //else
```

```
    bitm = bitm_data.images(0)
```

```
end
```

```
//take care of mipmaps
```

```
if bitm_data.image_info(image_index).num_mipmaps > 0 then
```

```
    dim mip_index as integer = mip_popup.ListIndex - 1 //because at 0 is "main"
```

```
    if mip_index >= 0 then
```

```
        bitm = bitm.mip_maps(mip_index)
```

```
    end
```

```
end
```

```
//take care of layers
```

```
if bitm_data.image_info(image_index).type = 1 then
```

```
    dim layer_index as integer = layer_popup.ListIndex - 1
```

```
    if layer_index > -1 then
```

```
        bitm = bitm.layers(layer_index)
```

```
    end
```

```
end
```

```
if bitm.mask_flag then
```

```
    //has an alpha channel
```

```
    mask_well.Enabled = true
```

```
    mask_text.Enabled = true
```

```
    main_well.Image = scaleit_max(bitm.image, 170)
```

```
    mask_well.Image = scaleit_max(bitm.mask, 170)
```

```
    main_full = bitm.image
```

```
    mask_full = bitm.mask
```

```
else
```

```

    //no alpha
    mask_well.Enabled = false
    mask_text.Enabled = false
    main_well.Image = scaleit_max(bitm.image, 170)
    main_full = bitm.image
end
End Sub
End Class

```

## **Class bitm\_preview\_window**

Inherits Window

### **bitm\_preview\_window.CloseWindow:**

Function CloseWindow() As Boolean

```

    me.close
    Return True

```

End Function

End Class

## **Class no\_meta\_control**

Inherits ContainerControl

End Class

## **Module H1SupportMap**

Global Enum MapTypeEnum As Integer

bitmaps = 1

sounds = 2

loc = 3

End Enum

Class SupportMap

### **SupportMap.BuildOnMap:**

```

Sub BuildOnMap()
    Dim bs as BinaryStream
    dim f as FolderItem = GetSaveFolderItem("map", MapType + "_new")
    dim StartTick, Endtick as integer
    StartTick = ticks
    if f <> nil then
        Dim NewPathOffset, NewIndexOffset as integer
        dim Index_paths() as integer
        Dim Index_offsets() as integer
        Dim Index_sizes() as integer

        bs = f.CreateBinaryFile("map")
        bs.LittleEndian = true

        'writes the map type

```

```

bs.Position = 0
if MapType = "bitmaps" then
    bs.WriteInt32(1)
elseif MapType = "sounds" then
    bs.WriteInt32(2)
elseif MapType = "loc" then
    bs.WriteInt32(3)
else
    bs.writeint32(-1)
end

'Writes the tag count
bs.Position = &hc

bs.WriteInt32(IndexCount + NewTagCount)

' writes default tags
if TagCache.Ubound <> -1 then
    For i as integer = 0 to TagCache.Ubound
        dim TempSup as new SupportData
        TempSup = TagCache(i)

        Index_offsets.Append(bs.Length)
        Index_sizes.Append(TempSup.Size)

        bs.Position = bs.Length
        Bs.write(TempSup.Data)

    next
end

If NewTags.Ubound <> -1 then
    For i as integer = 0 to NewTags.Ubound
        dim TempSup as SupportData
        TempSup = NewTags(i)

        Index_offsets.Append(bs.Length)
        Index_sizes.Append(TempSup.Size)

        bs.Position = Index_offsets(i)
        Bs.write(TempSup.Data)
    next
end

'path offset
NewPathOffset = bs.Length
bs.Position = &h4
bs.WriteInt32(NewPathOffset)

if TagCache.Ubound <> -1 then
    For i as integer = 0 to TagCache.Ubound
        dim TempSup as new SupportData
        TempSup = TagCache(i)

```



```

        Index_paths.Append(bs.Length - NewPathOffset)
        bs.Position = bs.Length

        bs.Write(TempSup.Path)
        bs.writeint8(0)
    next
end

If NewTags.Ubound <> -1 then
    For i as integer = 0 to NewTags.Ubound
        dim TempSup as SupportData
        TempSup = NewTags(i)

        Index_paths.Append(bs.Length - NewPathOffset)
        bs.Position = bs.Length

        bs.Write(TempSup.Path)
        bs.writeint8(0)
    next
end

NewIndexOffset = bs.Length
'index offset
bs.Position = &h8
bs.WriteInt32(NewIndexOffset)

for i as integer = 0 to Index_paths.Ubound
    bs.Position = bs.Length
    bs.WriteInt32(Index_paths(i))
    bs.WriteInt32(Index_sizes(i))
    bs.WriteInt32(Index_offsets(i))
next

bs.close
end
Endtick = ticks
MsgBox( "It took " + str((Endtick - StartTick) / 60) + " seconds to build the new file")
End Sub

```

### **SupportMap.CreateNewMap:**

```

Sub CreateNewMap()
    Dim bs as BinaryStream
    dim f as FolderItem = GetSaveFolderItem("map", MapType + "_new")
    dim StartTick, Endtick as integer
    StartTick = ticks
    if f <> nil then
        Dim NewPathOffset, NewIndexOffset as integer
        dim Index_paths() as integer
        Dim Index_offsets() as integer
        Dim Index_sizes() as integer

        bs = f.CreateBinaryFile("map")
        bs.LittleEndian = true
    end if
End Sub

```

```

'writes the map type
bs.Position = 0
if MapType = "bitmaps" then
    bs.WriteInt32(1)
elseif MapType = "sounds" then
    bs.WriteInt32(2)
elseif MapType = "loc" then
    bs.WriteInt32(3)
else
    bs.writeint32(-1)
end

```

```

'Writes the tag count
bs.Position = &hc

```

```

bs.WriteInt32(NewTagCount)

```

```

If NewTags.Ubound <> -1 then
    For i as integer = 0 to NewTags.Ubound
        dim TempSup as SupportData
        TempSup = NewTags(i)

        Index_offsets.Append(bs.Length)
        Index_sizes.Append(TempSup.Size)

        bs.Position = Index_offsets(i)
        Bs.write(TempSup.Data)
    next
end

```

```

'path offset
NewPathOffset = bs.Length
bs.Position = &h4
bs.WriteInt32(NewPathOffset)

```

```

If NewTags.Ubound <> -1 then
    For i as integer = 0 to NewTags.Ubound
        dim TempSup as SupportData
        TempSup = NewTags(i)

        Index_paths.Append(bs.Length - NewPathOffset)
        bs.Position = bs.Length

        bs.Write(TempSup.Path)
        bs.writeint8(0)
    next
end

```

```

NewIndexOffset = bs.Length
'index offset
bs.Position = &h8
bs.WriteInt32(NewIndexOffset)

```

```

    for i as integer = 0 to Index_paths.Ubound
        bs.Position = bs.Length
        bs.WriteInt32(Index_paths(i))
        bs.WriteInt32(Index_sizes(i))
        bs.WriteInt32(Index_offsets(i))
    next

    bs.close
end
Endtick = ticks
MsgBox( "It took " + str((Endtick - StartTick) / 60) + " seconds to build the new file")
End Sub

```

### SupportMap.inject:

Function inject(RawLoad as memoryBlock, Name as string) As integer

```

    if f = nil then return - 1
    dim bs as binaryStream = f.OpenAsBinaryFile(true)
    Dim Index, Raw, Strings as MemoryBlock
    Dim LocPathOffset, LocIndexOffset, LocIndexCount as integer
    Dim NewInd as new SupportData

```

```

    bs.LittleEndian = true

```

```

'finds out map type
dim Mem as integer
bs.Position = 0
mem = bs.ReadInt32
if mem = 1 then
    MapType = "bitmaps"
elseif mem = 2 then
    MapType = "sounds"
elseif mem = 3 then
    MapType = "loc"
end

```

```

'loads in header
LocPathOffset = bs.ReadInt32
LocIndexOffset = bs.ReadInt32
LocIndexCount = bs.ReadInt32

```

```

'loads in all raw data
Raw = bs.Read(LocPathOffset - 16)
'loads in all strings
bs.Position = LocPathOffset
Strings = bs.Read(LocIndexOffset - LocPathOffset)
'loads in all of index
bs.Position = LocIndexOffset
Index = bs.Read(Bs.Length - LocIndexOffset)

```

```

'fills out supportdata for index use
NewInd.Offset = LocPathOffset
NewInd.Size = RawLoad.Size

```

```

'writes new raw
bs.Position = LocPathOffset
bs.Write(RawLoad)

'string info
bs.Position = LocPathOffset + RawLoad.Size
LocPathOffset = bs.Position
bs.Write(Strings)
NewInd.PathOffset = bs.Position
bs.write("Eschaton\" + name + "_0:" + str(indexCount + 1))
bs.Writeint8(0)
LocIndexOffset = bs.Position

bs.Write(Index)
bs.Writeint32(NewInd.PathOffset)
bs.Writeint32(NewInd.Size)
bs.Writeint32(NewInd.Offset)

bs.Position = 4
bs.WriteInt32(LocPathOffset)
bs.WriteInt32(LocIndexOffset)
bs.WriteInt32(LocIndexCount + 1)

bs.Close
Return NewInd.Offset
End Function

```

### **SupportMap.read:**

```

Sub read(fs as folderItem)
    if fs = nil then Return
    f = fs
    if f <> nil then
        dim bs as binaryStream = f.OpenAsBinaryFile
        dim NewDic as new Dictionary

        bs.LittleEndian = true

        'finds out map type
        dim Mem as integer
        bs.Position = 0
        mem = bs.ReadInt32
        if mem = 1 then
            MapType = "bitmaps"
        elseif mem = 2 then
            MapType = "sounds"
        elseif mem = 3 then
            MapType = "loc"
        end

        'loads in header
        PathOffset = bs.ReadInt32
        IndexOffset = bs.ReadInt32
        IndexCount = bs.ReadInt32
    end if
End Sub

```

```

'reads index and produces data
For i as Integer = 0 to IndexCount - 1
    Dim TempSup as new SupportData
    bs.Position = IndexOffset + (i * 12)
    TempSup.PathOffset = bs.ReadInt32 + PathOffset
    TempSup.Size = bs.ReadInt32
    TempSup.Offset = bs.ReadInt32

    bs.Position = TempSup.PathOffset
    dim checker as integer = 1
    dim temp_str as string
    while checker <> 0
        temp_str = temp_str + bs.read(1)
        checker = bs.readint8
        bs.Position = bs.Position - 1
    Wend
    TempSup.Path = temp_str

'reads in the raw data
bs.Position = TempSup.Offset
TempSup.Data = bs.read(TempSup.size)
NewDic.Value(TempSup.Path) = i
TagCache.Append(TempSup)
next

bs.Close

TagDic = NewDic
end
End Sub

```

### **SupportMap.SpliceNewData:**

```

Function SpliceNewData(File as folderItem) As string
    dim bs as BinaryStream = File.OpenAsBinaryFile
    Dim TempSup as new SupportData
    Dim StrArray() as string

    If MapType = "bitmaps" then
        bs.Position = &h80
        TempSup.Data = bs.read(bs.Length - &h80)
        TempSup.Size = Bs.Length - &h80
        StrArray = split(Replace(str(File.AbsolutePath), MainDirectory.AbsolutePath, ""), ".dds")
        TempSup.path = StrArray(0)
    end

    NewTags.Append(TempSup)
    NewTagDic.Value(TempSup.Path) = NewTagCount
    Return TempSup.path
    bs.Close

    NewTagCount = NewTagCount + 1
End Function

```

### **SupportMap.SpliceNoString:**

```

Sub SpliceNoString(File as folderItem, Top as string)
    dim bs as BinaryStream = File.OpenAsBinaryFile
    Dim TempSup as new SupportData
    Dim StrArray() as string

    bs.Position = &h80
    TempSup.Data = bs.read(bs.Length - &h80)
    TempSup.Size = Bs.Length - &h80
    StrArray = split(Replace(str(File.AbsolutePath), Top, ""), ".dds")
    TempSup.path = StrArray(0)

    NewTags.Append(TempSup)
    NewTagDic.Value(TempSup.Path) = NewTagCount
    bs.Close

    NewTagCount = NewTagCount + 1
End Sub

```

### **SupportMap.UpdateNewTagDic:**

```

Sub UpdateNewTagDic()
    dim NDic as new Dictionary
    for i as integer = 0 to NewTags.Ubound
        Ndic.Value(NewTags(i).Path) = i
    next

    NewTagDic = NDic
End Sub

```

### **SupportMap.UpdateTagDic:**

```

Sub UpdateTagDic()
    dim NDic as new Dictionary
    for i as integer = 0 to TagCache.Ubound
        Ndic.Value(TagCache(i).Path) = i
    next

    TagDic = NDic
End Sub
f As folderItem

```

IndexCount As Integer

IndexOffset As Integer

MainDirectory As folderItem

MapType As String

NewTagCount As Integer

NewTagDic As dictionary

NewTags() As Supportdata

PathOffset As Integer

TagCache() As Supportdata

TagDic As dictionary

End Class

Class SupportData

Data As memoryBlock

Offset As Integer

Path As String

PathOffset As Integer

Size As Integer

End Class

End Module

## **Class main\_control\_RWC**

Inherits ContainerControl

### **main\_control\_RWC.click:**

Sub click(row as integer)

    dim temp\_str as string = Listbox1.CellTag(row, 0)

    dim str\_ar() as string = temp\_str.split(":")

    dim current\_plugin as new PluginLibUniversal

    current\_plugin = plugin\_info

    dim end\_type as string

    dim end\_index as string

    dim chunk\_number as integer

    dim name\_offset as integer

    dim br as BinaryStream = fs.OpenAsBinaryFile

    br.LittleEndian = true

    dim local\_offset as integer = offset

    dim control\_width as integer = me.Canvas1.Width

    dim control\_height as integer = me.Canvas1.Height

    //find the correct reflexive

    for i as integer = 0 to UBound(str\_ar)

        dim temp\_ar() as string = str\_ar(i).split("\_")

        if temp\_ar.Ubound < 1 then Return //in case some shennangins take place

        end\_type = temp\_ar(0)

        end\_index = temp\_ar(1)

        dim old\_end\_type as string

        dim old\_end\_index as string

```

if i > 0 then
    dim temp_ar_old() as string = str_ar(i-1).split("_")
    if temp_ar_old.ubound < 1 then Return //in case some shennangins take place
    old_end_type = temp_ar_old(0)
    old_end_index = temp_ar_old(1)
end
if end_type = "chunk" and IsNumeric(end_index) then
    br.Position = local_offset + current_plugin.reflexive_offset(val(old_end_index))
    dim chunk_count as integer = br.ReadInt32
    dim ref_offset as integer = br.ReadInt32 - magic
    dim chunk_size as integer = current_plugin.reflexives(val(old_end_index)).this_reflexive_size
    chunk_number = val(end_index)
    local_offset = ref_offset + chunk_size*chunk_number
    name_offset = current_plugin.reflexive_name_offset(val(old_end_index))
    current_plugin = current_plugin.reflexives(val(old_end_index))
end
next

if IsNumeric(end_index) then
    //item
    select case end_type
        //add the correct RWC editor control
    case "bitmask16"
        dim temp_ints(-1) as Integer
        for i as integer = 0 to UBound(current_plugin.bitmask16_data(val(end_index)).values)
            temp_ints.Append( current_plugin.bitmask16_data(val(end_index)).values(i) )
        next
        dim edit_offset as integer = local_offset + current_plugin.bitmask16_offset(val(end_index))
        dim temp_bitmask as new bitmask16_editor_control_RWC(fs, edit_offset,
            current_plugin.bitmask16_name(val(end_index)), _
            current_plugin.bitmask16_data(val(end_index)).labels, temp_ints)
        temp_bitmask.name_text.Text = current_plugin.bitmask16_name(val(end_index))
        if embedded <> nil then
            embedded.Close
        end
        temp_bitmask.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
        temp_bitmask.refresh
        temp_bitmask.read
        embedded = temp_bitmask
    case "bitmask32"
        dim temp_ints(-1) as Integer
        for i as integer = 0 to UBound(current_plugin.bitmask32_data(val(end_index)).values)
            temp_ints.Append( current_plugin.bitmask32_data(val(end_index)).values(i) )
        next
        dim edit_offset as integer = local_offset + current_plugin.bitmask32_offset(val(end_index))
        dim temp_bitmask as new bitmask32_editor_control_RWC(fs, edit_offset,
            current_plugin.bitmask32_name(val(end_index)), _
            current_plugin.bitmask32_data(val(end_index)).labels, temp_ints)
        temp_bitmask.name_text.Text = current_plugin.bitmask32_name(val(end_index))
        if embedded <> nil then
            embedded.Close
        end
        temp_bitmask.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
        temp_bitmask.refresh
    end
end

```



```

temp_bitmask.read
embedded = temp_bitmask
case "bitmask8"
    dim temp_ints(-1) as Integer
    for i as integer = 0 to UBound(current_plugin.bitmask8_data(val(end_index)).values)
        temp_ints.Append( current_plugin.bitmask8_data(val(end_index)).values(i) )
    next
    dim edit_offset as integer = local_offset + current_plugin.bitmask8_offset(val(end_index))
    dim temp_bitmask as new bitmask8_editor_control_RWC(fs, edit_offset,
        current_plugin.bitmask8_name(val(end_index)), _
        current_plugin.bitmask8_data(val(end_index)).labels, temp_ints)
    temp_bitmask.name_text.Text = current_plugin.bitmask8_name(val(end_index))
    if embedded <> nil then
        embedded.Close
    end
    temp_bitmask.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_bitmask.refresh
    temp_bitmask.read
    embedded = temp_bitmask
case "colorARGB"
    dim edit_offset as integer = local_offset + current_plugin.colorARGB_offset(val(end_index))
    dim temp_color as new colorARGB_editor_control_RWC(fs, edit_offset,
        current_plugin.colorARGB_name(val(end_index)))
    if embedded <> nil then
        embedded.Close
    end
    temp_color.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_color.Refresh
    temp_color.read
    embedded = temp_color
case "colorbyte"
    dim edit_offset as integer = local_offset + current_plugin.colorbyte_offset(val(end_index))
    dim temp_color as new colorbyte_editor_control_RWC(fs, edit_offset,
        current_plugin.colorbyte_name(val(end_index)))
    if embedded <> nil then
        embedded.Close
    end
    temp_color.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_color.Refresh
    temp_color.read
    embedded = temp_color
case "colorRGB"
    dim edit_offset as integer = local_offset + current_plugin.colorRGB_offset(val(end_index))
    dim temp_color as new colorRGB_editor_control_RWC(fs, edit_offset,
        current_plugin.colorRGB_name(val(end_index)))
    if embedded <> nil then
        embedded.Close
    end
    temp_color.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_color.Refresh
    temp_color.read
    embedded = temp_color
case "dependency"
    dim edit_offset as integer = local_offset + current_plugin.dependency_offset(val(end_index))

```

```

dim temp_dep as new dependency_editor_control_RWC(fs, edit_offset,
current_plugin.dependency_name(val(end_index)), map)
if embedded <> nil then
    embedded.Close
end
temp_dep.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
temp_dep.Refresh
temp_dep.read
embedded = temp_dep
case "double"
    dim edit_offset as integer = local_offset + current_plugin.double_offset(val(end_index))
    dim temp_double as new double_editor_control_RWC(fs, edit_offset,
current_plugin.double_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_double.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_double.Refresh
    temp_double.read
    embedded = temp_double
case "float"
    dim edit_offset as integer = local_offset + current_plugin.float_offset(val(end_index))
    dim temp_float as new float_editor_control_RWC(fs, edit_offset, current_plugin.float_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_float.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_float.Refresh
    temp_float.read
    embedded = temp_float
case "id16"
    dim edit_offset as integer = local_offset + current_plugin.id16_offset(val(end_index))
    dim temp_ints(-1) as Int16
    for i as integer = 0 to UBound(current_plugin.id16_data(val(end_index)).values)
        temp_ints.Append( current_plugin.id16_data(val(end_index)).values(i) )
    next
    dim temp_enum as new id16_editor_control_RWC(fs, edit_offset, current_plugin.id16_name(val(end_index)),
current_plugin.id16_data(val(end_index)).labels, temp_ints)
    if embedded <> nil then
        embedded.Close
    end
    temp_enum.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_enum.Refresh
    temp_enum.read
    embedded = temp_enum
case "id32"
    dim edit_offset as integer = local_offset + current_plugin.id32_offset(val(end_index))
    dim temp_ints(-1) as Int32
    for i as integer = 0 to UBound(current_plugin.id32_data(val(end_index)).values)
        temp_ints.Append( current_plugin.id32_data(val(end_index)).values(i) )
    next
    dim temp_enum as new id32_editor_control_RWC(fs, edit_offset, current_plugin.id32_name(val(end_index)),
current_plugin.id32_data(val(end_index)).labels, temp_ints)
    if embedded <> nil then

```

```

        embedded.Close
    end
    temp_enum.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_enum.Refresh
    temp_enum.read
    embedded = temp_enum
case "id8"
    dim edit_offset as integer = local_offset + current_plugin.id8_offset(val(end_index))
    dim temp_ints(-1) as Int8
    for i as integer = 0 to UBound(current_plugin.id8_data(val(end_index)).values)
        temp_ints.Append( current_plugin.id8_data(val(end_index)).values(i) )
    next
    dim temp_enum as new id8_editor_control_RWC(fs, edit_offset, current_plugin.id8_name(val(end_index)),
    current_plugin.id8_data(val(end_index)).labels, temp_ints)
    if embedded <> nil then
        embedded.Close
    end
    temp_enum.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_enum.Refresh
    temp_enum.read
    embedded = temp_enum
case "index"
    dim edit_offset as integer = local_offset + current_plugin.index_offset(val(end_index))
    dim temp_ind as new index_editor_control_RWC(fs, offset, edit_offset,
    current_plugin.index_name(val(end_index)), _
    current_plugin.index_data(val(end_index)), temp_str, plugin_info, magic)
    if embedded <> nil then
        embedded.Close
    end
    temp_ind.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_ind.Refresh
    temp_ind.read
    embedded = temp_ind
case "int16"
    dim edit_offset as integer = local_offset + current_plugin.int16_offset(val(end_index))
    dim temp_int16 as new int16_editor_control_RWC(fs, edit_offset, current_plugin.int16_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_int16.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_int16.Refresh
    temp_int16.read
    embedded = temp_int16
case "int32"
    dim edit_offset as integer = local_offset + current_plugin.int32_offset(val(end_index))
    dim temp_int32 as new int32_editor_control_RWC(fs, edit_offset, current_plugin.int32_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_int32.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_int32.Refresh
    temp_int32.read
    embedded = temp_int32
case "int8"

```

```

dim edit_offset as integer = local_offset + current_plugin.int8_offset(val(end_index))
dim temp_int8 as new int8_editor_control_RWC(fs, edit_offset, current_plugin.int8_name(val(end_index)) )
if embedded <> nil then
    embedded.Close
end
temp_int8.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
temp_int8.Refresh
temp_int8.read
embedded = temp_int8
case "lonelD"
    dim edit_offset as integer = local_offset + current_plugin.lonelD_offset(val(end_index))
    dim temp_dep as new lonelD_editor_control_RWC(fs, edit_offset, current_plugin.lonelD_name(val(end_index)),
    map)
    if embedded <> nil then
        embedded.Close
    end
    temp_dep.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_dep.Refresh
    temp_dep.read
    embedded = temp_dep
case "string128"
    dim edit_offset as integer = local_offset + current_plugin.string128_offset(val(end_index))
    dim temp_string128 as new string128_editor_control_RWC(fs, edit_offset,
    current_plugin.string128_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_string128.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_string128.Refresh
    temp_string128.read
    embedded = temp_string128
case "string32"
    dim edit_offset as integer = local_offset + current_plugin.string32_offset(val(end_index))
    dim temp_string32 as new string32_editor_control_RWC(fs, edit_offset,
    current_plugin.string32_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_string32.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_string32.Refresh
    temp_string32.read
    embedded = temp_string32
case "string4"
    dim edit_offset as integer = local_offset + current_plugin.string4_offset(val(end_index))
    dim temp_string4 as new string4_editor_control_RWC(fs, edit_offset,
    current_plugin.string4_name(val(end_index)) )
    if embedded <> nil then
        embedded.Close
    end
    temp_string4.EmbedWithin(me.canvas1, 0, 0, control_width, control_height)
    temp_string4.Refresh
    temp_string4.read
    embedded = temp_string4
end select

```

```
end
End Sub
```

### **main\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_magic as integer)
    fs = f
    offset = in_offset
    magic = in_magic //for passing along to reflexive editor controls
    plugin_info = new PluginLibUniversal
End Sub
```

### **main\_control\_RWC.Destructor:**

```
Sub Destructor()
    if Embedded <> nil then
        embedded.Close
    end
End Sub
```

### **main\_control\_RWC.init:**

```
Sub init(f_xml as folderItem, in_entity_style as boolean, in_show_hidden as boolean, byref in_map as H1.map)
    map = in_map
    entity_style = in_entity_style
    show_hidden = in_show_hidden
    load(f_xml)
End Sub
```

### **main\_control\_RWC.load:**

```
Sub load(f_xml as folderItem)
    dim ref(-1) as integer
    if Embedded <> nil then
        embedded.Close
    end

    //plugin related init
    change_ok = false
    dim f_top as FolderItem = GetFolderItem("").Child("Plugins")
    dim class_str as string = f_xml.name.mid(1,4)
    PopupMenu1.DeleteAllRows
    for i as integer = 1 to f_top.Count
        dim f_xml_folder as FolderItem = f_top.item(i)
        if f_xml_folder.exists and f_xml_folder.Directory then
            if f_xml_folder.Child(class_str + ".ent").Exists then
                dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".ent")
                PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
                PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
            end
            if f_xml_folder.Child(class_str + ".xml").Exists then
                dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".xml")
                PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
                PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
            end
        end
    end
end
```

```

next
for i as integer = 0 to PopupMenu1.ListCount - 1
    if PopupMenu1.RowTag(i) = f_xml.Parent.Name + ":" + f_xml.Name then
        PopupMenu1.ListIndex = i
        exit for i
    end
next
change_ok = true

if entity_style then
    plugin_info.refresh_as_Ent(f_xml, false, ref, show_hidden)
else
    plugin_info.refresh_as_HMT(f_xml, false, ref)
end
//end plugin related stuff

ListBox1.DeleteAllRows
if ubound(plugin_info.bitmask16_offset) > -1 then
    ListBox1.AddFolder("Bitmask16s")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "bitmask16_f"
end
if ubound(plugin_info.bitmask32_offset) > -1 then
    ListBox1.AddFolder("Bitmask32s")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "bitmask32_f"
end
if ubound(plugin_info.bitmask8_offset) > -1 then
    ListBox1.AddFolder("Bitmask8s")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "bitmask8_f"
end
if ubound(plugin_info.colorARGB_offset) > -1 then
    ListBox1.AddFolder("ColorARGBs")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "colorARGB_f"
end
if ubound(plugin_info.colorbyte_offset) > -1 then
    ListBox1.AddFolder("Colorbytes")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "colorbyte_f"
end
if ubound(plugin_info.colorRGB_offset) > -1 then
    ListBox1.AddFolder("ColorRGBs")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "colorRGB_f"
end
if ubound(plugin_info.dependency_offset) > -1 then
    ListBox1.AddFolder("Dependencies")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "dependency_f"
end
if ubound(plugin_info.double_offset) > -1 then
    ListBox1.AddFolder("Doubles")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "double_f"
end
if ubound(plugin_info.float_offset) > -1 then
    ListBox1.AddFolder("Floats")
    listBox1.CellTag(Listbox1.LastIndex, 0) = "float_f"
end
if ubound(plugin_info.id16_offset) > -1 then

```

```

        Listbox1.AddFolder("Enum16s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "id16_f"
    end
    if ubound(plugin_info.id32_offset) > -1 then
        Listbox1.AddFolder("Enum32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "id32_f"
    end
    if ubound(plugin_info.id8_offset) > -1 then
        Listbox1.AddFolder("Enum8s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "id8_f"
    end
    if ubound(plugin_info.index_offset) > -1 then
        Listbox1.AddFolder("Indexes")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "index_f"
    end
    if ubound(plugin_info.int16_offset) > -1 then
        Listbox1.AddFolder("Int16s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "int16_f"
    end
    if ubound(plugin_info.int32_offset) > -1 then
        Listbox1.AddFolder("Int32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "int32_f"
    end
    if ubound(plugin_info.int8_offset) > -1 then
        Listbox1.AddFolder("Int8s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "int8_f"
    end
    if ubound(plugin_info.lonelD_offset) > -1 then
        Listbox1.AddFolder("LoneIDs")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "lonelD_f"
    end
    if ubound(plugin_info.reflexive_offset) > -1 then
        Listbox1.AddFolder("Reflexives")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "reflexive_f"
    end
    if ubound(plugin_info.string128_offset) > -1 then
        Listbox1.AddFolder("String128s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "string128_f"
    end
    if ubound(plugin_info.string32_offset) > -1 then
        Listbox1.AddFolder("String32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "string32_f"
    end
    if ubound(plugin_info.string4_offset) > -1 then
        Listbox1.AddFolder("String4s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = "string4_f"
    end
end
End Sub
change_ok As boolean

embedded As ContainerControl

entity_style As boolean

```

fs As folderItem

magic As Integer

map As h1.map

offset As Integer

plugin\_info As PluginLibUniversal

show\_hidden As boolean

### **main\_control\_RWC Note: folder info**

folder info

0 based index

for main, straight data type and index example float number 3

float\_2

for reflexives

reflexive\_<reflexiveindex>:chunk\_<chunknumber>:colorRGB\_0

for folders

bitmask32\_f

main>reflexives>reflexive\_i>

chunk1

chunk2

chunk3

### **main\_control\_RWC Control ListBox1:**

Sub ExpandRow(row As Integer)

dim temp\_str as string = ListBox1.CellTag(row, 0)

dim str\_ar() as string = temp\_str.split(":")

dim current\_plugin as new PluginLibUniversal

current\_plugin = plugin\_info

dim end\_type as string

dim end\_index as string

dim chunk\_number as integer

dim name\_offset as integer

dim br as BinaryStream = fs.OpenAsBinaryFile

br.LittleEndian = true

dim local\_offset as integer = offset

//find the correct reflexive

for i as integer = 0 to UBound(str\_ar)

dim temp\_ar() as string = str\_ar(i).split("\_")

if temp\_ar.Ubound < 1 then Return //in case some shennangins take place

end\_type = temp\_ar(0)

end\_index = temp\_ar(1)



```

dim old_end_type as string
dim old_end_index as string
if i > 0 then
    dim temp_ar_old() as string = str_ar(i-1).split("_")
    if temp_ar_old.ubound < 1 then Return //in case some shennangins take place
    old_end_type = temp_ar_old(0)
    old_end_index = temp_ar_old(1)
end
if end_type = "chunk" and IsNumeric(end_index) then
    br.Position = local_offset + current_plugin.reflexive_offset(val(old_end_index))
    dim chunk_count as integer = br.ReadInt32
    dim ref_offset as integer = br.ReadInt32 - magic
    dim chunk_size as integer = current_plugin.reflexives(val(old_end_index)).this_reflexive_size
    chunk_number = val(end_index)
    local_offset = ref_offset + chunk_size*chunk_number
    name_offset = current_plugin.reflexive_name_offset(val(old_end_index))
    current_plugin = current_plugin.reflexives(val(old_end_index))
end
next

select case end_type
case "reflexive"

    if IsNumeric(end_index) then
        //display all chunks
        br.Position = local_offset + current_plugin.reflexive_offset(val(end_index))
        name_offset = current_plugin.reflexive_name_offset(val(end_index))
        dim chunk_count as integer = br.ReadInt32
        dim ref_offset as integer = br.ReadInt32 - magic
        dim chunk_size as integer = current_plugin.reflexives(val(end_index)).this_reflexive_size
        if ref_offset < offset then chunk_count = -1
        for i as integer = 0 to chunk_count-1

            if name_offset > -1 then
                //unique names for chunks
                br.Position = ref_offset + i*chunk_size + name_offset
                dim name_str as string = br.read(32)
                name_str = name_str.replaceAll(chr(0),"")
                ListBox1.AddFolder(name_str)
                ListBox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "chunk_" + str(i)
            else
                //use reflexive names instead
                dim num_str as string = str(i+1)
                dim num_size as integer = len(str(chunk_count+1))
                while len(num_str) < num_size
                    num_str = "0" + num_str
                wend
                ListBox1.AddFolder(current_plugin.reflexive_name(val(end_index)) + " " + num_str)
                ListBox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "chunk_" + str(i)
            end

        next

    else

```

```

//display all reflexives
for i as integer = 0 to UBound(current_plugin.reflexive_name)
    br.Position = local_offset + current_plugin.reflexive_offset(i)
    dim count as integer = br.ReadInt32
    if count > 0 then
        ListBox1.AddFolder(current_plugin.reflexive_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "reflexive_" + str(i)
    end
end
next

end

case "chunk"
//display all possible data types within a given chunk

if ubound(current_plugin.bitmask16_offset) > -1 then
    ListBox1.AddFolder("Bitmask16s")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "bitmask16_f"
end
if ubound(current_plugin.bitmask32_offset) > -1 then
    ListBox1.AddFolder("Bitmask32s")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "bitmask32_f"
end
if ubound(current_plugin.bitmask8_offset) > -1 then
    ListBox1.AddFolder("Bitmask8s")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "bitmask8_f"
end
if ubound(current_plugin.colorARGB_offset) > -1 then
    ListBox1.AddFolder("ColorARGBs")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "colorARGB_f"
end
if ubound(current_plugin.colorbyte_offset) > -1 then
    ListBox1.AddFolder("Colorbytes")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "colorbyte_f"
end
if ubound(current_plugin.colorRGB_offset) > -1 then
    ListBox1.AddFolder("ColorRGBs")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "colorRGB_f"
end
if ubound(current_plugin.dependency_offset) > -1 then
    ListBox1.AddFolder("Dependencies")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "dependency_f"
end
if ubound(current_plugin.double_offset) > -1 then
    ListBox1.AddFolder("Doubles")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "double_f"
end
if ubound(current_plugin.float_offset) > -1 then
    ListBox1.AddFolder("Floats")
    listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "float_f"
end
if ubound(current_plugin.id16_offset) > -1 then
    ListBox1.AddFolder("Enum16s")

```

```

        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "id16_f"
    end
    if ubound(current_plugin.id32_offset) > -1 then
        Listbox1.AddFolder("Enum32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "id32_f"
    end
    if ubound(current_plugin.id8_offset) > -1 then
        Listbox1.AddFolder("Enum8s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "id8_f"
    end
    if ubound(current_plugin.index_offset) > -1 then
        Listbox1.AddFolder("Indexes")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "index_f"
    end
    if ubound(current_plugin.int16_offset) > -1 then
        Listbox1.AddFolder("Int16s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "int16_f"
    end
    if ubound(current_plugin.int32_offset) > -1 then
        Listbox1.AddFolder("Int32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "int32_f"
    end
    if ubound(current_plugin.int8_offset) > -1 then
        Listbox1.AddFolder("Int8s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "int8_f"
    end
    if ubound(current_plugin.loneID_offset) > -1 then
        ListBox1.AddFolder("LoneIDs")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "loneID_f"
    end
    if ubound(current_plugin.reflexive_offset) > -1 then
        Listbox1.AddFolder("Reflexives")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "reflexive_f"
    end
    if ubound(current_plugin.string128_offset) > -1 then
        Listbox1.AddFolder("String128s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "string128_f"
    end
    if ubound(current_plugin.string32_offset) > -1 then
        Listbox1.AddFolder("String32s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "string32_f"
    end
    if ubound(current_plugin.string4_offset) > -1 then
        Listbox1.AddFolder("String4s")
        listbox1.CellTag(Listbox1.LastIndex, 0) = temp_str + ":" + "string4_f"
    end
end

case else
    //determine whether it's a folder or an item
    if not IsNumeric(end_index) then
        //folder
        select case end_type
            //add all of the items of that type
            case "bitmask16"

```

```

    for i as integer = 0 to UBound(current_plugin.bitmask16_name)
        ListBox1.AddRow(current_plugin.bitmask16_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "bitmask16_" + str(i)
    next
case "bitmask32"
    for i as integer = 0 to UBound(current_plugin.bitmask32_name)
        ListBox1.AddRow(current_plugin.bitmask32_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "bitmask32_" + str(i)
    next
case "bitmask8"
    for i as integer = 0 to UBound(current_plugin.bitmask8_name)
        ListBox1.AddRow(current_plugin.bitmask8_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "bitmask8_" + str(i)
    next
case "colorARGB"
    for i as integer = 0 to UBound(current_plugin.colorARGB_name)
        ListBox1.AddRow(current_plugin.colorARGB_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "colorARGB_" + str(i)
    next
case "colorbyte"
    for i as integer = 0 to UBound(current_plugin.colorbyte_name)
        ListBox1.AddRow(current_plugin.colorbyte_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "colorbyte_" + str(i)
    next
case "colorRGB"
    for i as integer = 0 to UBound(current_plugin.colorRGB_name)
        ListBox1.AddRow(current_plugin.colorRGB_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "colorRGB_" + str(i)
    next
case "dependency"
    for i as integer = 0 to UBound(current_plugin.dependency_name)
        ListBox1.AddRow(current_plugin.dependency_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "dependency_" + str(i)
    next
case "double"
    for i as integer = 0 to UBound(current_plugin.double_name)
        ListBox1.AddRow(current_plugin.double_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "double_" + str(i)
    next
case "float"
    for i as integer = 0 to UBound(current_plugin.float_name)
        ListBox1.AddRow(current_plugin.float_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "float_" + str(i)
    next
case "id16"
    for i as integer = 0 to UBound(current_plugin.id16_name)
        ListBox1.AddRow(current_plugin.id16_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "id16_" + str(i)
    next
case "id32"
    for i as integer = 0 to UBound(current_plugin.id32_name)
        ListBox1.AddRow(current_plugin.id32_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "id32_" + str(i)
    next

```

```

case "id8"
    for i as integer = 0 to UBound(current_plugin.id8_name)
        ListBox1.AddRow(current_plugin.id8_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "id8_" + str(i)
    next
case "index"
    for i as integer = 0 to UBound(current_plugin.index_name)
        ListBox1.AddRow(current_plugin.index_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "index_" + str(i)
    next
case "int16"
    for i as integer = 0 to UBound(current_plugin.int16_name)
        ListBox1.AddRow(current_plugin.int16_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "int16_" + str(i)
    next
case "int32"
    for i as integer = 0 to UBound(current_plugin.int32_name)
        ListBox1.AddRow(current_plugin.int32_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "int32_" + str(i)
    next
case "int8"
    for i as integer = 0 to UBound(current_plugin.int8_name)
        ListBox1.AddRow(current_plugin.int8_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "int8_" + str(i)
    next
case "lonelD"
    for i as integer = 0 to UBound(current_plugin.lonelD_name)
        ListBox1.AddRow(current_plugin.lonelD_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "lonelD_" + str(i)
    next
case "string128"
    for i as integer = 0 to UBound(current_plugin.string128_name)
        ListBox1.AddRow(current_plugin.string128_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "string128_" + str(i)
    next
case "string32"
    for i as integer = 0 to UBound(current_plugin.string32_name)
        ListBox1.AddRow(current_plugin.string32_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "string32_" + str(i)
    next
case "string4"
    for i as integer = 0 to UBound(current_plugin.string4_name)
        ListBox1.AddRow(current_plugin.string4_name(i))
        ListBox1.CellTag(ListBox1.LastIndex, 0) = temp_str + ":" + "string4_" + str(i)
    next
end select
end
end select

br.close
End Sub

```

```

Function Click(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    click(row)

```

End Function

Function KeyDown(Key As String) As Boolean

```
//31 for down 30 for up
dim row as integer
dim column as integer = 0
dim ok as boolean = false
select case asc(key)
case 31
    if me.ListIndex + 1 < me.listcount then
        row = me.ListIndex + 1
        ok = true
    end
case 30
    if me.ListIndex - 1 > -1 then
        row = me.ListIndex - 1
        ok = true
    end
end
```

```
if ok then
    click(row)
end
```

End Function

### **main\_control\_RWC Control PopupMenu1:**

Sub Change()

```
if not change_ok then return
dim temp_strs() as string = split(me.RowTag(me.ListIndex), ":")
if temp_strs.ubound <> 1 then return
dim folder as FolderItem = GetFolderItem("").Child("Plugins").Child(temp_strs(0))
if Folder.Exists and Folder.Directory then
    dim plugin as FolderItem = Folder.Child(temp_strs(1))
    if plugin.Exists then
        dim temp_split() as string = plugin.name.split(".")
        dim temp_str as string
        if temp_split.Ubound < 1 then
            'dim temp_split2() as string = plugin.AbsolutePath.Split(":")
            'dim temp_str2 as string = temp_split2(temp_split2.Ubound)
            'dim temp_split3() as string = temp_str2.split(".")
            'if temp_split3.Ubound < 1 then
            'break
            'errorbox("There was an error determining plugin extension type")
            'return
            'end
            'dim temp_str3 as string = temp_split3(1)
            '#if buildtargetwindows
            'temp_str3 = temp_str3.left(temp_str3.length - 1)
            '#endif
            'temp_str = temp_str3
            break
            return
        else
            temp_str = temp_split(1)
```

```

        end
        select case temp_str.Uppercase
        case "ENT"
            entity_style = true
        case "XML"
            entity_style = false
        end select
        load(plugin)
    end
end
End Sub
End Class

```

## **Class bitmask16\_editor\_control\_RWC**

Inherits ContainerControl

### **bitmask16\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset
    name_text.Text = in_name
    EditField1.Text = hex(offset)
    labels = in_labels
    values = in_values
End Sub

```

### **bitmask16\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt16
    br.Close

    Listbox1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        Listbox1.AddRow(labels(i))
        ListBox1.CellTag(i,0) = i
    next
    Listbox1.ColumnType(0) = 2

    dim temp_str as string = bin(value)
    while temp_str.len < 16
        temp_str = "0" + temp_str
    wend
    for i as integer = 0 to Listbox1.ListCount - 1
        if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
            Listbox1.CellCheck(i,0) = true
        else
            Listbox1.CellCheck(i,0) = false
        end
    next
end

```

End Sub

### **bitmask16\_editor\_control\_RWC.update\_value:**

```
Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 16)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
    for i as integer = 0 to ListBox1.ListCount - 1
        if ListBox1.CellCheck(i,0) then
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        else
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        end
    next

    value = val("&b" + temp_str)
End Sub
```

### **bitmask16\_editor\_control\_RWC.write:**

```
Sub write()
    update_value

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt16(value)
    bw.Close
End Sub
fs As folderItem
```

### **bitmask16\_editor\_control\_RWC.labels():**

```
labels() As string
//the label
offset As Integer

value As int16
```

### **bitmask16\_editor\_control\_RWC.values():**

```
values() As Integer
//the i-th bit
```

### **bitmask16\_editor\_control\_RWC Control ListBox1:**

```
Sub CellAction(row As Integer, column As Integer)
    //this is where to check to see if they checked a box
    update_value
    write
```



```
End Sub
End Class
```

## **Class bitmask32\_editor\_control\_RWC**

Inherits ContainerControl

### **bitmask32\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset
    name_text.Text = in_name
    EditField1.Text = hex(offset)
    labels = in_labels
    values = in_values
End Sub
```

### **bitmask32\_editor\_control\_RWC.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.Readuint32
    br.Close

    Listbox1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        Listbox1.AddRow(labels(i))
        ListBox1.CellTag(i,0) = i
    next
    Listbox1.ColumnType(0) = 2

    dim temp_str as string = bin(value)
    while temp_str.len < 32
        temp_str = "0" + temp_str
    wend
    for i as integer = 0 to Listbox1.ListCount - 1
        if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
            Listbox1.CellCheck(i,0) = true
        else
            Listbox1.CellCheck(i,0) = false
        end
    next
End Sub
```

### **bitmask32\_editor\_control\_RWC.update\_value:**

```
Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 32)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
```

```

for i as integer = 0 to Listbox1.ListCount - 1
    //dim front_side as string = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1)
    //dim back_side as string = temp_str.mid(values(Listbox1.CellTag(i,0))+1)
    if ListBox1.CellCheck(i,0) then
        temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
    else
        temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
    end
next

value = val("&b" + temp_str)
End Sub

```

### **bitmask32\_editor\_control\_RWC.write:**

```

Sub write()
    //update_value

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.Writeuint32(value)
    bw.Close
End Sub
fs As folderItem

```

### **bitmask32\_editor\_control\_RWC.labels():**

```

labels() As string
//the label
offset As Integer

```

```

value As uint32

```

### **bitmask32\_editor\_control\_RWC.values():**

```

values() As Integer
//the i-th bit

```

### **bitmask32\_editor\_control\_RWC Control ListBox1:**

```

Sub CellAction(row As Integer, column As Integer)
    //this is where to check to see if they checked a box
    update_value
    write
End Sub
End Class

```

## **Class bitmask8\_editor\_control\_RWC**

```

Inherits ContainerControl

```

### **bitmask8\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as integer)
    fs = f
    offset = in_offset
    name_text.Text = in_name
    EditField1.Text = hex(offset)
    labels = in_labels
    values = in_values
End Sub

```

### **bitmask8\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt8
    br.Close

    Listbox1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        Listbox1.AddRow(labels(i))
        ListBox1.CellTag(i,0) = i
    next
    Listbox1.ColumnType(0) = 2

    dim temp_str as string = bin(value)
    while temp_str.len < 8
        temp_str = "0" + temp_str
    wend
    for i as integer = 0 to Listbox1.ListCount - 1
        if temp_str.mid(values(Listbox1.CellTag(i,0)), 1) = "1" then
            Listbox1.CellCheck(i,0) = true
        else
            Listbox1.CellCheck(i,0) = false
        end
    next
End Sub

```

### **bitmask8\_editor\_control\_RWC.update\_value:**

```

Sub update_value()
    dim temp_str as string = bin(value)
    while(temp_str.Len < 8)
        temp_str = "0" + temp_str
    wend

    //fix the string interpretation
    for i as integer = 0 to Listbox1.ListCount - 1
        if ListBox1.CellCheck(i,0) then
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "1" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        else
            temp_str = temp_str.mid(1, values(Listbox1.CellTag(i,0))-1) + "0" + temp_str.mid(values(Listbox1.CellTag(i,0))+1)
        end
    next

```

```

    value = val("&b" + temp_str)
End Sub

```

### **bitmask8\_editor\_control\_RWC.write:**

```

Sub write()
    update_value

    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt8(value)
    bw.Close
End Sub
fs As folderItem

```

### **bitmask8\_editor\_control\_RWC.labels():**

```

labels() As string
//the label
offset As Integer

```

```

value As int8

```

### **bitmask8\_editor\_control\_RWC.values():**

```

values() As Integer
//the i-th bit

```

### **bitmask8\_editor\_control\_RWC Control ListBox1:**

```

Sub CellAction(row As Integer, column As Integer)
    //this is where to check to see if they checked a box
    update_value
    write
End Sub
End Class

```

## **Class colorARGB\_editor\_control\_RWC**

Inherits ContainerControl

### **colorARGB\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.Text = hex(in_offset)
    name_text.Text = in_name
    change_ok = false
End Sub

```

### **colorARGB\_editor\_control\_RWC.read:**

```

Sub read()

```

```

dim br as BinaryStream = fs.OpenAsBinaryFile
br.LittleEndian = true
br.Position = offset

alpha = br.ReadSingle
red = br.ReadSingle
green = br.ReadSingle
blue = br.ReadSingle

br.Close

Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
change_ok = false
alpha_slider.Value = alpha*100
alpha_edit.Text = str(alpha_slider.Value)
red_slider.Value = red*100
red_edit.Text = str(red_slider.Value)
green_slider.Value = green*100
green_edit.Text = str(green_slider.Value)
blue_slider.Value = blue*100
blue_edit.Text = str(blue_slider.Value)
change_ok = true
End Sub

```

#### **colorARGB\_editor\_control\_RWC.update\_value:**

```

Sub update_value()
    alpha = alpha_slider.Value/100.0
    alpha_edit.Text = str(alpha_slider.Value)
    red = red_slider.Value/100.0
    red_edit.Text = str(red_slider.Value)
    green = green_slider.Value/100.0
    green_edit.Text = str(green_slider.Value)
    blue = blue_slider.Value/100.0
    blue_edit.Text = str(blue_slider.Value)
    Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
End Sub

```

#### **colorARGB\_editor\_control\_RWC.write:**

```

Sub write()
    update_value
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = Offset
    bw.WriteSingle(alpha)
    bw.WriteSingle(red)
    bw.WriteSingle(green)
    bw.WriteSingle(blue)
    bw.close
End Sub

```

alpha As single

blue As single

change\_ok As boolean

fs As folderItem

green As single

offset As Integer

red As single

### **colorARGB\_editor\_control\_RWC Control BevelButton1:**

Sub Action()

if not change\_ok then return

dim temp\_color as color = RGB(red \* 255, green \* 255, blue \* 255)

if SelectColor(temp\_color, "Select a Color") then

dim R as single = temp\_color.Red

dim G as single = temp\_color.Green

dim B as single = temp\_color.Blue

red = R/255.0

green = G/255.0

blue = B/255.0

Rectangle1.FillColor = RGB(red \* 255, green \* 255, blue \* 255)

change\_ok = false

alpha\_slider.Value = alpha\*100

alpha\_edit.Text = str(alpha\_slider.Value)

red\_slider.Value = red\*100

red\_edit.Text = str(red\_slider.Value)

green\_slider.Value = green\*100

green\_edit.Text = str(green\_slider.Value)

blue\_slider.Value = blue\*100

blue\_edit.Text = str(blue\_slider.Value)

change\_ok = true

update\_value

write

end

End Sub

### **colorARGB\_editor\_control\_RWC Control alpha\_slider:**

Sub ValueChanged()

if not change\_ok then return

update\_value

write

End Sub

### **colorARGB\_editor\_control\_RWC Control red\_slider:**

Sub ValueChanged()

if not change\_ok then return

update\_value

write

End Sub

### **colorARGB\_editor\_control\_RWC Control green\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

```

### **colorARGB\_editor\_control\_RWC Control blue\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub
End Class

```

## **Class colorRGB\_editor\_control\_RWC**

Inherits ContainerControl

### **colorRGB\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.Text = hex(in_offset)
    name_text.Text = in_name
    change_ok = false
End Sub

```

### **colorRGB\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset

    red = br.ReadSingle
    green = br.ReadSingle
    blue = br.ReadSingle

    br.Close

    Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
    change_ok = false
    red_slider.Value = red*100
    red_edit.Text = str(red_slider.Value)
    green_slider.Value = green*100
    green_edit.Text = str(green_slider.Value)
    blue_slider.Value = blue*100
    blue_edit.Text = str(blue_slider.Value)
    change_ok = true
End Sub

```

### **colorRGB\_editor\_control\_RWC.update\_value:**

```

Sub update_value()
    red = red_slider.Value/100.0

```

```

red_edit.Text = str(red_slider.Value)
green = green_slider.Value/100.0
green_edit.Text = str(green_slider.Value)
blue = blue_slider.Value/100.0
blue_edit.Text = str(blue_slider.Value)
Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
End Sub

```

### **colorRGB\_editor\_control\_RWC.write:**

```

Sub write()
    update_value
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = Offset
    bw.WriteSingle(red)
    bw.WriteSingle(green)
    bw.WriteSingle(blue)
    bw.close
End Sub
blue As single

```

change\_ok As boolean

fs As folderItem

green As single

offset As Integer

red As single

### **colorRGB\_editor\_control\_RWC Control BevelButton1:**

```

Sub Action()
    if not change_ok then return
    dim temp_color as color = RGB(red * 255, green * 255, blue * 255)

    if SelectColor(temp_color, "Select a Color") then
        dim R as single = temp_color.Red
        dim G as single = temp_color.Green
        dim B as single = temp_color.Blue
        red = R/255.0
        green = G/255.0
        blue = B/255.0
        Rectangle1.FillColor = RGB(red * 255, green * 255, blue * 255)
        change_ok = false
        red_slider.Value = red*100
        red_edit.Text = str(red_slider.Value)
        green_slider.Value = green*100
        green_edit.Text = str(green_slider.Value)
        blue_slider.Value = blue*100
        blue_edit.Text = str(blue_slider.Value)
        change_ok = true
        update_value
    end if
End Sub

```



```

        write
    end
End Sub

colorRGB_editor_control_RWC Control red_slider:

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

colorRGB_editor_control_RWC Control green_slider:

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

colorRGB_editor_control_RWC Control blue_slider:

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub
End Class

```

## **Class colorbyte\_editor\_control\_RWC**

Inherits ContainerControl

### **colorbyte\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.Text = hex(in_offset)
    name_text.Text = in_name
    change_ok = false
End Sub

```

### **colorbyte\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset

    blue = br.ReadUInt8
    green = br.ReadUInt8
    red = br.ReadUInt8
    alpha = br.ReadUInt8

    br.Close

```

```

Rectangle1.FillColor = RGB(red, green, blue)
change_ok = false
alpha_slider.Value = (alpha/255.0)*100
alpha_edit.Text = str(alpha_slider.Value)
red_slider.Value = (red/255.0)*100
red_edit.Text = str(red_slider.Value)
green_slider.Value = (green/255.0)*100
green_edit.Text = str(green_slider.Value)
blue_slider.Value = (blue/255.0)*100
blue_edit.Text = str(blue_slider.Value)
change_ok = true
End Sub

```

### **colorbyte\_editor\_control\_RWC.update\_value:**

```

Sub update_value()
    alpha = (alpha_slider.Value/100.0)*255
    alpha_edit.Text = str(alpha_slider.Value)
    red = (red_slider.Value/100.0)*255
    red_edit.Text = str(red_slider.Value)
    green = (green_slider.Value/100.0)*255
    green_edit.Text = str(green_slider.Value)
    blue = (blue_slider.Value/100.0)*255
    blue_edit.Text = str(blue_slider.Value)
    Rectangle1.FillColor = RGB(red, green, blue)
End Sub

```

### **colorbyte\_editor\_control\_RWC.write:**

```

Sub write()
    update_value
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = Offset
    bw.Writeuint8(blue)
    bw.Writeuint8(green)
    bw.Writeuint8(red)
    bw.Writeuint8(alpha)
    bw.close
End Sub

```

alpha As single

blue As single

change\_ok As boolean

fs As folderItem

green As single

offset As Integer

red As single

### **colorbyte\_editor\_control\_RWC Control BevelButton1:**

```

Sub Action()
    if not change_ok then return
    dim temp_color as color = RGB(red, green, blue)

    if SelectColor(temp_color, "Select a Color") then
        dim R as single = temp_color.Red
        dim G as single = temp_color.Green
        dim B as single = temp_color.Blue
        red = R
        green = G
        blue = B
        Rectangle1.FillColor = RGB(red, green, blue)
        change_ok = false
        alpha_slider.Value = (alpha/255.0)*100
        alpha_edit.Text = str(alpha_slider.Value)
        red_slider.Value = (red/255.0)*100
        red_edit.Text = str(red_slider.Value)
        green_slider.Value = (green/255.0)*100
        green_edit.Text = str(green_slider.Value)
        blue_slider.Value = (blue/255.0)*100
        blue_edit.Text = str(blue_slider.Value)
        change_ok = true
        update_value
        write
    end
End Sub

```

#### **colorbyte\_editor\_control\_RWC Control alpha\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

```

#### **colorbyte\_editor\_control\_RWC Control red\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

```

#### **colorbyte\_editor\_control\_RWC Control green\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value
    write
End Sub

```

#### **colorbyte\_editor\_control\_RWC Control blue\_slider:**

```

Sub ValueChanged()
    if not change_ok then return
    update_value

```

```
write
End Sub
End Class
```

## **Class dependency\_editor\_control\_RWC**

Inherits ContainerControl

### **dependency\_editor\_control\_RWC.class\_changed:**

```
Sub class_changed()
    if change_ok then
        dim class_index as integer = PopupMenu1.ListIndex
        change_ok = false
        PopupMenu2.DeleteAllRows
        PopupMenu2.AddRow("nulled out")
        PopupMenu2.RowTag(0) = &hFFFFFFF
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
        next
        PopupMenu2.ListIndex = -1
        change_ok = true
    end
End Sub
```

### **dependency\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, byref in_map as H1.map)
    fs = f
    offset = in_offset
    offset_edit.Text = hex(in_offset)
    name_text.Text = in_name
    map = in_map
    change_ok = false
End Sub
```

### **dependency\_editor\_control\_RWC.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value_class_str = br.Read(4)
    value_name_offset = br.ReadInt32 - map.magic
    br.Position = br.Position + 4
    value_ID = br.ReadUInt32
    if value_ID <> &hFFFFFFF then
```

```

    if map.tagIDTable.haskey(value_ID) then
        value_class_str = map.tags(map.tagIDTable.value(value_ID)).class1
        value_name_offset = map.tags(map.tagIDTable.value(value_ID)).nameoffset
    end
end
br.Close

update
End Sub

```

### **dependency\_editor\_control\_RWC.update:**

```

Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    dim class_index as integer = -1
    for i as integer = 0 to UBound(map.meta_class_folders.child)
        dim temp_str() as string = split(map.meta_class_folders.child(i).name, ":")
        dim temp_class1_str as string = temp_str(0)
        PopupMenu1.AddRow(temp_class1_str)
        if value_class_str = reverse(temp_class1_str) then
            PopupMenu1.ListIndex = i
            class_index = i
        end
    next
    if class_index = -1 then
        PopupMenu1.ListIndex = -1
    end
    PopupMenu2.DeleteAllRows
    PopupMenu2.AddRow("nulled out")
    PopupMenu2.RowTag(0) = &hFFFFFFF
    if value_ID = &hFFFFFFF then
        PopupMenu2.ListIndex = 0
    end
    if class_index > -1 then
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
            if value_ID = map.tags(indexs(i)).tagID then
                PopupMenu2.ListIndex = i+1
            end
        next
    end
    change_ok = true
    me.Refresh
End Sub

```

dependency\_editor\_control\_RWC.write:

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset  
    bw.Write(value_class_str)  
    bw.WriteInt32(value_name_offset + map.magic)  
    bw.WriteInt32(0)  
    bw.WriteUInt32(value_ID)  
    bw.Close  
End Sub  
change_ok As boolean
```

fs As folderItem

map As H1.map

offset As Integer

value\_class\_str As string

value\_ID As uint32

value\_name\_offset As Integer

### **dependency\_editor\_control\_RWC Control PopupMenu1:**

```
Sub Change()  
    if change_ok then  
        class_changed  
    end  
End Sub
```

### **dependency\_editor\_control\_RWC Control PopupMenu2:**

```
Sub Change()  
    if change_ok then  
        value_ID = PopupMenu2.RowTag(PopupMenu2.ListIndex)  
        if value_ID <> &hFFFFFF then  
            if map.tagIDtable.haskey(value_ID) then  
                value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1  
                value_name_offset = map.tags(map.tagIDtable.value(value_ID)).nameoffset  
            end  
        end  
        write  
    end  
End Sub  
End Class
```

## **Class double\_editor\_control\_RWC**

Inherits ContainerControl

### **double\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.text = in_name
End Sub

```

### **double\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.Readdouble
    EditField1.Text = str(value)
    br.Close
End Sub

```

### **double\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.Writedouble(value)
    bw.Close
End Sub

```

fs As folderItem

offset As Integer

value As double

### **double\_editor\_control\_RWC Control EditField1:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class

```

## **Class float\_editor\_control\_RWC**

Inherits ContainerControl

### **float\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.text = in_name
End Sub

```

### **float\_editor\_control\_RWC.read:**

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset  
    value = br.ReadSingle  
    EditField1.Text = str(value)  
    br.Close  
End Sub
```

### **float\_editor\_control\_RWC.write:**

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset  
    bw.Writesingle(value)  
    bw.Close  
End Sub  
fs As folderItem
```

offset As Integer

value As single

### **float\_editor\_control\_RWC Control EditField1:**

```
Function KeyDown(Key As String) As Boolean  
    if asc(key) = 3 or asc(key) = 13 then  
        value = val(EditField1.text)  
        write  
        return true  
    end  
End Function  
End Class
```

## **Class id8\_editor\_control\_RWC**

Inherits ContainerControl

### **id8\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as int8)  
    fs = f  
    offset = in_offset  
    offset_edit.Text = hex(in_offset)  
    name_text.Text = in_name  
    labels = in_labels  
    values = in_values  
    change_ok = false  
End Sub
```

### **id8\_editor\_control\_RWC.read:**



```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt8
    br.Close

```

```

    update

```

```

End Sub

```

### **id8\_editor\_control\_RWC.update:**

```

Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        PopupMenu1.AddRow(labels(i))
        if value = values(i) then
            PopupMenu1.ListIndex = i
        end
    next
    change_ok = true
End Sub

```

### **id8\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt8(value)
    bw.Close
End Sub

```

```

change_ok As boolean

```

```

fs As folderItem

```

```

labels() As string

```

```

offset As Integer

```

```

value As int8

```

```

values() As int8

```

### **id8\_editor\_control\_RWC Control PopupMenu1:**

```

Sub Change()
    if change_ok then
        value = values(me.ListIndex)
        write
    end
End Sub

```

```

End Sub

```

```

End Class

```

## **Class id16\_editor\_control\_RWC**

Inherits ContainerControl

### **id16\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as int16)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.text = in_name
    labels = in_labels
    values = in_values
    change_ok = false
End Sub
```

### **id16\_editor\_control\_RWC.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt16
    br.Close

    update
End Sub
```

### **id16\_editor\_control\_RWC.update:**

```
Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    for i as integer = 0 to UBound(labels)
        PopupMenu1.AddRow(labels(i))
        if value = values(i) then
            PopupMenu1.ListIndex = i
        end
    next
    change_ok = true
End Sub
```

### **id16\_editor\_control\_RWC.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt16(value)
    bw.Close
End Sub
change_ok As boolean
```

fs As folderItem

labels() As string

offset As Integer

value As int16

values() As int16

### **id16\_editor\_control\_RWC Control PopupMenu1:**

```
Sub Change()  
    if change_ok then  
        value = values(me.ListIndex)  
        write  
    end  
End Sub  
End Class
```

## **Class id32\_editor\_control\_RWC**

Inherits ContainerControl

### **id32\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, in_labels() as string, in_values() as integer)  
    fs = f  
    offset = in_offset  
    offset_edit.text = hex(in_offset)  
    name_text.Text = in_name  
    labels = in_labels  
    values = in_values  
    change_ok = false  
End Sub
```

### **id32\_editor\_control\_RWC.read:**

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset  
    value = br.ReadInt32  
    br.Close  
  
    update  
End Sub
```

### **id32\_editor\_control\_RWC.update:**

```
Sub update()  
    change_ok = false  
    PopupMenu1.DeleteAllRows  
    for i as integer = 0 to UBound(labels)  
        PopupMenu1.AddRow(labels(i))  
        if value = values(i) then  
            PopupMenu1.ListIndex = i  
        end if  
    next i  
end Sub
```

```

        end
    next
    change_ok = true
End Sub

```

### **id32\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt32(value)
    bw.Close
End Sub
change_ok As boolean

```

fs As folderItem

labels() As string

offset As Integer

value As int32

values() As int32

### **id32\_editor\_control\_RWC Control PopupMenu1:**

```

Sub Change()
    if change_ok then
        value = values(me.ListIndex)
        write
    end
End Sub
End Class

```

## **Class index\_editor\_control\_RWC**

Inherits ContainerControl

### **index\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_tag_offset as integer, in_offset as integer, in_name as string, in_path as string,
in_this_path as string, in_plugin_info as PluginLibUniversal, in_magic as integer)
    fs = f
    tag_offset = in_tag_offset
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.Text = in_name
    path = in_path
    path_text.text = path.ReplaceAll(":", ": ")
    this_path = in_this_path
    plugin_info = in_plugin_info
    magic = in_magic
    change_ok = false

```

```
load
End Sub
```

### **index\_editor\_control\_RWC.load:**

```
Private Sub load()
    change_ok = false
    PopupMenu1.DeleteAllRows
    PopupMenu1.AddRow("None Selected")
    PopupMenu1.RowTag(0) = -1

    //determine which chunks of upper level reflexives
    dim this_path_ar() as string = this_path.Split(":")
    dim path_indexs(-1) as integer
    path_indexs.append 0
    for i as integer = 0 to UBound(this_path_ar)
        if this_path_ar(i).instr("reflexive") > 0 then
            dim temp_str_ar() as string
            temp_str_ar = this_path_ar(i).split("_")
            if IsNumeric(temp_str_ar(temp_str_ar.ubound)) then
                path_indexs.append val(temp_str_ar(temp_str_ar.ubound))
            end
        end
    end
next

dim br as BinaryStream = fs.OpenAsBinaryFile
br.littleendian = true

//find the offset of the target reflexive
dim path_ar() as string = path.split(":")
dim current_plugin as PluginLibUniversal = plugin_info
dim max_count as integer = 0
dim chunk_size as integer = 0
dim name_offset as integer
dim target_offset as integer = tag_offset
for i as integer = 1 to UBound(path_ar)
    for j as integer = 0 to UBound(current_plugin.reflexives)
        if current_plugin.reflexive_name(j) = path_ar(i) then
            br.Position = target_offset + path_indexs(0) * chunk_size + current_plugin.reflexive_offset(j)
            path_indexs.remove(0)
            max_count = br.readint32 - 1
            target_offset = br.readint32 - magic
            name_offset = current_plugin.reflexive_name_offset(j)
            current_plugin = current_plugin.reflexives(j)
            chunk_size = current_plugin.this_reflexive_size
            exit for j
        end
    next
next

dim chunk_names(-1) as string
if name_offset <> -1 then
    for i as integer = 0 to max_count
        br.position = target_offset + (i*chunk_size)_
```

```

        + name_offset
        dim temp_str as string = br.read(32)
        temp_str = temp_str.replaceAll(chr(0),"")
        chunk_names.append temp_str
    next
end
if name_offset <> -1 _
    and max_count <> 0 then
    for i as integer = 0 to max_count
        PopupMenu1.AddRow(chunk_names(i))
        PopupMenu1.RowTag(i+1) = i
    next
else
    for i as integer = 0 to max_count
        PopupMenu1.AddRow("Chunk " + str(i+1))
        PopupMenu1.RowTag(i+1) = i
    next
end
change_ok = true
br.Close
End Sub

```

### **index\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt16
    change_ok = false
    PopupMenu1.ListIndex = value + 1
    change_ok = true
    br.Close
End Sub

```

### **index\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt16(value)
    bw.Close
End Sub
change_ok As boolean

```

fs As folderItem

magic As Integer

offset As Integer

path As string

plugin\_info As PluginLibUniversal

tag\_offset As Integer

this\_path As string

value As int16

### **index\_editor\_control\_RWC Control PopupMenu1:**

```
Sub Change()  
    if change_ok then  
        value = me.RowTag(me.ListIndex)  
        write  
    end  
End Sub  
End Class
```

## **Class int8\_editor\_control\_RWC**

Inherits ContainerControl

### **int8\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string)  
    fs = f  
    offset = in_offset  
    offset_edit.text = hex(in_offset)  
    name_text.Text = in_name  
End Sub
```

### **int8\_editor\_control\_RWC.read:**

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset  
    value = br.ReadInt8  
    EditField1.Text = str(value)  
    br.Close  
End Sub
```

### **int8\_editor\_control\_RWC.write:**

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset  
    bw.WriteInt8(value)  
    bw.Close  
End Sub  
fs As folderItem
```

offset As Integer

value As Int8

int8\_editor\_control\_RWC Control EditField1:

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class
```

## **Class int16\_editor\_control\_RWC**

Inherits ContainerControl

### **int16\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.Text = in_name
End Sub
```

### **int16\_editor\_control\_RWC.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt16
    EditField1.Text = str(value)
    br.Close
End Sub
```

### **int16\_editor\_control\_RWC.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt16(value)
    bw.Close
End Sub
fs As folderItem
```

offset As Integer

value As Int16

### **int16\_editor\_control\_RWC Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
```



```

        write
    return true
end
End Function
End Class

```

## **Class int32\_editor\_control\_RWC**

Inherits ContainerControl

### **int32\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.Text = in_name
End Sub

```

### **int32\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.ReadInt32
    EditField1.Text = str(value)
    br.Close
End Sub

```

### **int32\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteInt32(value)
    bw.Close
End Sub
fs As folderItem

```

offset As Integer

value As Int32

### **int32\_editor\_control\_RWC Control EditField1:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        value = val(EditField1.text)
        write
        return true
    end
End Function
End Class

```

Class loneID\_editor\_control\_RWC

Inherits ContainerControl

### **loneID\_editor\_control\_RWC.class\_changed:**

```
Sub class_changed()
    if change_ok then
        dim class_index as integer = PopupMenu1.ListIndex
        change_ok = false
        PopupMenu2.DeleteAllRows
        PopupMenu2.AddRow("nulled out")
        PopupMenu2.RowTag(0) = &hFFFFFFF
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
        next
        PopupMenu2.ListIndex = -1
        change_ok = true
    end
End Sub
```

### **loneID\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string, byref in_map as H1.map)
    fs = f
    offset = in_offset
    offset_edit.Text = hex(in_offset)
    name_text.Text = in_name
    map = in_map
    change_ok = false
End Sub
```

### **loneID\_editor\_control\_RWC.read:**

```
Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value_ID = br.ReadUInt32
    value_class_str = "none"
    if value_ID <> &hFFFFFFF then
        if map.tagIDtable.haskey(value_ID) then
            value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
        end
    end
    br.Close
```

```
update
End Sub
```

### **lonID\_editor\_control\_RWC.update:**

```
Sub update()
    change_ok = false
    PopupMenu1.DeleteAllRows
    dim class_index as integer = -1
    for i as integer = 0 to UBound(map.meta_class_folders.child)
        dim temp_str() as string = split(map.meta_class_folders.child(i).name, ":")
        dim temp_class1_str as string = temp_str(0)
        PopupMenu1.AddRow(temp_class1_str)
        if value_class_str = reverse(temp_class1_str) then
            PopupMenu1.ListIndex = i
            class_index = i
        end
    next
    if class_index = -1 then
        PopupMenu1.ListIndex = -1
    end
    PopupMenu2.DeleteAllRows
    PopupMenu2.AddRow("nulled out")
    PopupMenu2.RowTag(0) = &hFFFFFFF
    if value_ID = &hFFFFFFF then
        PopupMenu2.ListIndex = 0
    end
    if class_index > -1 then
        dim indexs(-1) as integer
        dim strings(-1) as string
        for i as integer = 0 to UBound(map.meta_class_folders.child(class_index).item)
            dim index as integer = map.meta_class_folders.child(class_index).item(i)
            indexs.append index
            strings.append map.tags(index).name
        next
        strings.sortwith(indexs)
        for i as integer = 0 to UBound(indexs)
            PopupMenu2.AddRow(map.tags(indexs(i)).name)
            PopupMenu2.RowTag(i+1) = map.tags(indexs(i)).tagID
            if value_ID = map.tags(indexs(i)).tagID then
                PopupMenu2.ListIndex = i+1
            end
        next
    end
    change_ok = true
    me.Refresh
End Sub
```

### **lonID\_editor\_control\_RWC.write:**

```
Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.WriteUInt32(value_ID)
    bw.Close
End Sub
```

```
End Sub
change_ok As boolean
```

```
fs As folderItem
```

```
map As H1.map
```

```
offset As Integer
```

```
value_class_str As string
```

```
value_ID As uint32
```

### **loneID\_editor\_control\_RWC Control PopupMenu1:**

```
Sub Change()
    if change_ok then
        class_changed
    end
End Sub
```

### **loneID\_editor\_control\_RWC Control PopupMenu2:**

```
Sub Change()
    if change_ok then
        value_ID = PopupMenu2.RowTag(PopupMenu2.ListIndex)
        if value_ID <> &hFFFFFFFF then
            if map.tagIDtable.haskey(value_ID) then
                value_class_str = map.tags(map.tagIDtable.value(value_ID)).class1
            end
        end
        write
    end
End Sub
End Class
```

## **Class string4\_editor\_control\_RWC**

Inherits ContainerControl

### **string4\_editor\_control\_RWC.LostFocus:**

```
Sub LostFocus()
    me.read
End Sub
```

### **string4\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.text = in_name
End Sub
```

string4\_editor\_control\_RWC.read:

```
Sub read()  
    dim br as BinaryStream = fs.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset  
    value = br.Read(4)  
    EditField1.Text = value  
    br.Close  
End Sub
```

**string4\_editor\_control\_RWC.write:**

```
Sub write()  
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = offset  
    bw.Write(value)  
    bw.Close  
End Sub  
fs As folderItem
```

offset As Integer

value As string

**string4\_editor\_control\_RWC Control EditField1:**

```
Function KeyDown(Key As String) As Boolean  
    if asc(key) = 3 or asc(key) = 13 then  
        value = EditField1.text  
        value = value.LeftB(4)  
        while value.LenB < 4  
            value = value + chr(0)  
        wend  
        EditField1.Text = value  
        write  
        return true  
    end  
End Function
```

```
Sub LostFocus()  
    read  
End Sub
```

**string4\_editor\_control\_RWC Control PushButton1:**

```
Sub Action()  
    EditField1.text = EditField1.Text + chr(0)  
End Sub  
End Class
```

**Class string32\_editor\_control\_RWC**

Inherits ContainerControl

```
string32_editor_control_RWC.LostFocus:
```

```
Sub LostFocus()
```

```
    me.read
```

```
End Sub
```

### **string32\_editor\_control\_RWC.Constructor:**

```
Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
```

```
    fs = f
```

```
    offset = in_offset
```

```
    offset_edit.text = hex(in_offset)
```

```
    name_text.text = in_name
```

```
End Sub
```

### **string32\_editor\_control\_RWC.read:**

```
Sub read()
```

```
    dim br as BinaryStream = fs.OpenAsBinaryFile
```

```
    br.LittleEndian = true
```

```
    br.Position = offset
```

```
    value = br.Read(32)
```

```
    EditField1.Text = value
```

```
    br.Close
```

```
End Sub
```

### **string32\_editor\_control\_RWC.write:**

```
Sub write()
```

```
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
```

```
    bw.LittleEndian = true
```

```
    bw.Position = offset
```

```
    bw.Write(value)
```

```
    bw.Close
```

```
End Sub
```

```
fs As folderItem
```

```
offset As Integer
```

```
value As string
```

### **string32\_editor\_control\_RWC Control EditField1:**

```
Function KeyDown(Key As String) As Boolean
```

```
    if asc(key) = 3 or asc(key) = 13 then
```

```
        value = EditField1.text
```

```
        value = value.LeftB(32)
```

```
        while value.LenB < 32
```

```
            value = value + chr(0)
```

```
        wend
```

```
        EditField1.Text = value
```

```
        write
```

```
        return true
```

```
    end
```

```
End Function
```

```
Sub LostFocus()
```

```

    read
End Sub
string32_editor_control_RWC Control PushButton1:

```

```

Sub Action()
    EditField1.text = EditField1.Text + chr(0)
End Sub
End Class

```

## **Class string128\_editor\_control\_RWC**

Inherits ContainerControl

### **string128\_editor\_control\_RWC.LostFocus:**

```

Sub LostFocus()
    me.read
End Sub

```

### **string128\_editor\_control\_RWC.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_name as string)
    fs = f
    offset = in_offset
    offset_edit.text = hex(in_offset)
    name_text.text = in_name
End Sub

```

### **string128\_editor\_control\_RWC.read:**

```

Sub read()
    dim br as BinaryStream = fs.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    value = br.Read(128)
    EditField1.Text = value
    br.Close
End Sub

```

### **string128\_editor\_control\_RWC.write:**

```

Sub write()
    dim bw as BinaryStream = fs.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = offset
    bw.Write(value)
    bw.Close
End Sub

```

fs As folderItem

offset As Integer

value As string

### **string128\_editor\_control\_RWC Control EditField1:**

Function KeyDown(Key As String) As Boolean

if asc(key) = 3 or asc(key) = 13 then

value = EditField1.text

value = value.LeftB(128)

while value.LenB < 128

value = value + chr(0)

wend

EditField1.Text = value

write

return true

end

End Function

**string128\_editor\_control\_RWC Control PushButton1:**

Sub Action()

EditField1.text = EditField1.Text + chr(0)

End Sub

End Class

## **Class bitm\_import\_single\_RW**

Inherits Window

**bitm\_import\_single\_RW.CloseWindow:**

Function CloseWindow() As Boolean

self.Close

Return True

End Function

**bitm\_import\_single\_RW.Constructor:**

Sub Constructor()

// Calling the overridden superclass constructor.

Super.Window

info = new bitm\_image\_info

End Sub

**bitm\_import\_single\_RW.init:**

Sub init(input as bitm\_image\_info, in\_map\_f as folderItem, in\_bitmaps\_f as folderItem)

info = input

map\_f = in\_map\_f

bitmaps\_f = in\_bitmaps\_f

End Sub

bitmaps\_f As folderItem

info As bitm\_image\_info

map\_f As folderItem

target As picture

**bitm\_import\_single\_RW Control import\_push:**



```

Sub Action()
    dim data as MemoryBlock = create_bitmap(info, target)

    dim bw as BinaryStream
    dim ok as boolean = true
    if BitAnd(info.flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(ok)
    else
        bw = map_f.OpenAsBinaryFile(ok)
    end
    bw.LittleEndian = true
    bw.Position = info.offset
    bw.Write(data)
    bw.Close
    MsgBox("Bitmap Imported Successfully!")
    self.Close
End Sub

```

### **bitm\_import\_single\_RW Control load\_push:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target = temp_pic
    ImageWell1.Image = FileManip.scaleit_max(target, 255)
    import_push.Enabled = true
End Sub
End Class

```

## **Class bitm\_import\_single\_mask\_RW**

Inherits Window

### **bitm\_import\_single\_mask\_RW.CloseWindow:**

```

Function CloseWindow() As Boolean
    self.Close

    Return True

```

End Function

### **bitm\_import\_single\_mask\_RW.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window
    info = new bitm_image_info
End Sub

```

### **bitm\_import\_single\_mask\_RW.init:**

```

Sub init(input as bitm_image_info, in_map_f as folderItem, in_bitmaps_f as folderItem)
    info = input
    map_f = in_map_f
    bitmaps_f = in_bitmaps_f
End Sub
bitmaps_f As folderitem

```

info As bitm\_image\_info

map\_f As folderItem

target\_image As picture

target\_mask As picture

### **bitm\_import\_single\_mask\_RW Control load\_image\_push:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_image = temp_pic
    ImageWell_image.Image = FileManip.scaleit_max(target_image, 255)
    if target_image <> nil and target_mask <> nil then
        import_push.Enabled = true
    end
End Sub

```

### **bitm\_import\_single\_mask\_RW Control import\_push:**

```

Sub Action()
    dim data as MemoryBlock = create_bitmap(info, target_image, target_mask)

    dim bw as BinaryStream
    dim ok as boolean = true
    if BitAnd(info.flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(ok)
    end

```

```

else
    bw = map_f.OpenAsBinaryFile(ok)
end
bw.LittleEndian = true
bw.Position = info.offset
bw.Write(data)
bw.Close
MsgBox("Bitmap Imported Successfully!")
self.Close
End Sub

```

### **bitm\_import\_single\_mask\_RW Control load\_mask\_push:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_mask = temp_pic
    ImageWell_mask.Image = FileManip.scaleit_max(target_mask, 255)
    if target_image <> nil and target_mask <> nil then
        import_push.Enabled = true
    end
End Sub
End Class

```

## **Class bitm\_import\_cube\_RW**

Inherits Window

### **bitm\_import\_cube\_RW.check\_enabled:**

```

Sub check_enabled()
    if target_posx <> nil AND target_posy <> nil AND target_posz <> nil AND _
        target_negx <> nil AND target_negy <> nil AND target_negz <> nil then
        import_push.Enabled = true
    end
End Sub

```

### **bitm\_import\_cube\_RW.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window
    info = new bitm_image_info
End Sub

```

```

bitm_import_cube_RW.init:
Sub init(input as bitm_image_info, in_map_f as folderItem, in_bitmaps_f as folderItem)
    info = input
    map_f = in_map_f
    bitmaps_f = in_bitmaps_f
End Sub
bitmaps_f As folderItem

info As bitm_image_info

map_f As folderItem

target_negx As picture

target_negy As picture

target_negz As picture

target_posx As picture

target_posy As picture

target_posz As picture

```

### **bitm\_import\_cube\_RW Control load\_posx:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_posx = temp_pic
    ImageWell_posx.Image = FileManip.scaleit_max(target_posx, 95)
    check_enabled
End Sub

```

### **bitm\_import\_cube\_RW Control load\_negx:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width

```

```

dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_negx = temp_pic
ImageWell_negx.Image = FileManip.scaleit_max(target_negx, 95)
check_enabled
End Sub

```

### **bitm\_import\_cube\_RW Control load\_posy:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_posy = temp_pic
    ImageWell_posy.Image = FileManip.scaleit_max(target_posy, 95)
    check_enabled
End Sub

```

### **bitm\_import\_cube\_RW Control load\_negy:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_negy = temp_pic
    ImageWell_negy.Image = FileManip.scaleit_max(target_negy, 95)
    check_enabled
End Sub

```

### **bitm\_import\_cube\_RW Control load\_posz:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_posz = temp_pic
ImageWell_posz.Image = FileManip.scaleit_max(target_posz, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_RW Control load\_negz:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_negz = temp_pic
ImageWell_negz.Image = FileManip.scaleit_max(target_negz, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_RW Control import\_push:**

Sub Action()

```
dim targets(-1) as Picture = Array(target_posx, target_negx, target_posy, target_negy, target_posz, target_negz)
dim data as MemoryBlock = create_bitmaps(info, targets)

dim bw as BinaryStream
dim ok as boolean = true
if BitAnd(info.flags, &h100) = &h100 then
    bw = bitmaps_f.OpenAsBinaryFile(ok)
else
    bw = map_f.OpenAsBinaryFile(ok)
end
bw.LittleEndian = true
bw.Position = info.offset
bw.Write(data)
```

```

    bw.Close
    MsgBox("Cubemap Imported Successfully!")
    self.Close
End Sub
End Class

```

## **Class bitm\_import\_cube\_mask\_RW**

Inherits Window

### **bitm\_import\_cube\_mask\_RW.check\_enabled:**

```

Sub check_enabled()
    if target_posx <> nil AND target_posy <> nil AND target_posz <> nil AND _
        target_negx <> nil AND target_negy <> nil AND target_negz <> nil AND _
        target_posx_mask <> nil AND target_posy_mask <> nil AND target_posz_mask <> nil AND _
        target_negx_mask <> nil AND target_negy_mask <> nil AND target_negz_mask <> nil then
        import_push.Enabled = true
    end
End Sub

```

### **bitm\_import\_cube\_mask\_RW.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window
    info = new bitm_image_info
End Sub

```

### **bitm\_import\_cube\_mask\_RW.init:**

```

Sub init(input as bitm_image_info, in_map_f as folderItem, in_bitmaps_f as folderItem)
    info = input
    map_f = in_map_f
    bitmaps_f = in_bitmaps_f
End Sub

bitmaps_f As folderItem

info As bitm_image_info

map_f As folderItem

target_negx As picture

target_negx_mask As picture

target_negy As picture

target_negy_mask As picture

target_negz As picture

target_negz_mask As picture

target_posx As picture

```

target\_posx\_mask As picture

target\_posy As picture

target\_posy\_mask As picture

target\_posz As picture

target\_posz\_mask As picture

### **bitm\_import\_cube\_mask\_RW Control load\_posx:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_posx = temp_pic
ImageWell_posx.Image = FileManip.scaleit_max(target_posx, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_mask\_RW Control load\_negx:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_negx = temp_pic
ImageWell_negx.Image = FileManip.scaleit_max(target_negx, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_mask\_RW Control load\_posy:**

Sub Action()



```

dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_posy = temp_pic
ImageWell_posy.Image = FileManip.scaleit_max(target_posy, 95)
check_enabled
End Sub

```

### **bitm\_import\_cube\_mask\_RW Control load\_negy:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_negy = temp_pic
    ImageWell_negy.Image = FileManip.scaleit_max(target_negy, 95)
    check_enabled
End Sub

```

### **bitm\_import\_cube\_mask\_RW Control load\_posz:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_posz = temp_pic

```

```

ImageWell_posz.Image = FileManip.scaleit_max(target_posz, 95)
check_enabled
End Sub

```

### **bitm\_import\_cube\_mask\_RW Control load\_negz:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_negz = temp_pic
    ImageWell_negz.Image = FileManip.scaleit_max(target_negz, 95)
    check_enabled
End Sub

```

### **bitm\_import\_cube\_mask\_RW Control import\_push:**

```

Sub Action()
    dim targets(-1) as Picture = Array(target_posx, target_negx, target_posy, target_negy, target_posz, target_negz)
    dim masks(-1) as Picture = Array(target_posx_mask, target_negx_mask, target_posy_mask, target_negy_mask,
    target_posz_mask, target_negz_mask)
    dim data as MemoryBlock = create_bitmaps(info, targets, masks)

    dim bw as BinaryStream
    dim ok as boolean = true
    if BitAnd(info.flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(ok)
    else
        bw = map_f.OpenAsBinaryFile(ok)
    end
    bw.LittleEndian = true
    bw.Position = info.offset
    bw.Write(data)
    bw.Close
    MsgBox("Cubemap Imported Successfully!")
    self.Close
End Sub

```

### **bitm\_import\_cube\_mask\_RW Control load\_posx\_mask:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture

```

```

dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_posx_mask = temp_pic
ImageWell_posx_mask.Image = FileManip.scaleit_max(target_posx_mask, 95)
check_enabled
End Sub

```

#### **bitm\_import\_cube\_mask\_RW Control load\_negx\_mask:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_negx_mask = temp_pic
    ImageWell_negx_mask.Image = FileManip.scaleit_max(target_negx_mask, 95)
    check_enabled
End Sub

```

#### **bitm\_import\_cube\_mask\_RW Control load\_posy\_mask:**

```

Sub Action()
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_posy_mask = temp_pic
    ImageWell_posy_mask.Image = FileManip.scaleit_max(target_posy_mask, 95)
    check_enabled
End Sub

```

#### **bitm\_import\_cube\_mask\_RW Control load\_negy\_mask:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_negy_mask = temp_pic
ImageWell_negy_mask.Image = FileManip.scaleit_max(target_negy_mask, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_mask\_RW Control load\_posz\_mask:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
target_posz_mask = temp_pic
ImageWell_posz_mask.Image = FileManip.scaleit_max(target_posz_mask, 95)
check_enabled
```

End Sub

### **bitm\_import\_cube\_mask\_RW Control load\_negz\_mask:**

Sub Action()

```
dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
if f = nil then Return
if not f.Exists then Return
if f.OpenAsPicture = nil then Return
dim temp_pic as Picture = f.OpenAsPicture
dim temp_width as double = info.width
dim temp_height as double = info.height
dim width_scale as double = temp_pic.Width / temp_width
dim height_scale as double = temp_pic.Height / temp_height
if width_scale <> height_scale then
    errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
    return
end
```

```

end
target_negz_mask = temp_pic
ImageWell_negz_mask.Image = FileManip.scaleit_max(target_negz_mask, 95)
check_enabled
End Sub
End Class

```

## **Class bitm\_import\_volume\_RW**

Inherits Window

### **bitm\_import\_volume\_RW.CloseWindow:**

```

Function CloseWindow() As Boolean
    self.Close
    Return True

```

End Function

### **bitm\_import\_volume\_RW.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window
    info = new bitm_image_info
End Sub

```

### **bitm\_import\_volume\_RW.init:**

```

Sub init(input as bitm_image_info, in_map_f as folderItem, in_bitmaps_f as folderItem)
    info = input
    map_f = in_map_f
    bitmaps_f = in_bitmaps_f

    //do more here
    redim targets(-1)
    if info.depth >= 1 then
        redim targets(info.depth - 1)
    else
        break
    end
    redim images(-1)
    change_ok = false
    volume_popup.DeleteAllRows
    for i as integer = 0 to info.depth - 1
        volume_popup.AddRow("Layer " + str(i+1))
        images.Append false
    next
    volume_popup.ListIndex = 0
    change_ok = true
End Sub
bitmaps_f As folderItem

```

change\_ok As boolean

images() As boolean

info As bitm\_image\_info

map\_f As folderItem

targets() As picture

### **bitm\_import\_volume\_RW Control import\_push:**

Sub Action()

dim data as MemoryBlock = create\_volume\_bitmaps(info, targets)

dim bw as BinaryStream

dim ok as boolean = true

if BitAnd(info.flags, &h100) = &h100 then

bw = bitmaps\_f.OpenAsBinaryFile(ok)

else

bw = map\_f.OpenAsBinaryFile(ok)

end

bw.LittleEndian = true

bw.Position = info.offset

bw.Write(data)

bw.Close

MsgBox("Bitmaps Imported Successfully!")

self.Close

End Sub

### **bitm\_import\_volume\_RW Control load\_push:**

Sub Action()

dim layer\_index as integer = volume\_popup.ListIndex

dim f as FolderItem = GetOpenFolderItem(image\_filetypes.All)

if f = nil then Return

if not f.Exists then Return

if f.OpenAsPicture = nil then Return

dim temp\_pic as Picture = f.OpenAsPicture

dim temp\_width as double = info.width

dim temp\_height as double = info.height

dim width\_scale as double = temp\_pic.Width / temp\_width

dim height\_scale as double = temp\_pic.Height / temp\_height

if width\_scale <> height\_scale then

errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")

return

end

targets(layer\_index) = temp\_pic

ImageWell1.Image = FileManip.scaleit\_max(targets(layer\_index), 255)

images(layer\_index) = true

dim import\_ok as Boolean = true

for i as integer = 0 to UBound(images)

if not images(i) then

import\_ok = false

exit for i

end

next

```

    if import_ok then
        import_push.Enabled = true
    end
End Sub

```

### **bitm\_import\_volume\_RW Control volume\_popup:**

```

Sub Change()
    if not change_ok then return

    dim layer_index as integer = me.ListIndex
    if images(layer_index) then
        ImageWell1.Image = FileManip.scaleit_max(targets(layer_index), 255)
    else
        ImageWell1.Image = nil
    end
End Sub
End Class

```

## **Class bitm\_import\_volume\_mask\_RW**

Inherits Window

### **bitm\_import\_volume\_mask\_RW.CloseWindow:**

```

Function CloseWindow() As Boolean
    self.Close

```

```

    Return True

```

```

End Function

```

### **bitm\_import\_volume\_mask\_RW.Constructor:**

```

Sub Constructor()
    // Calling the overridden superclass constructor.
    Super.Window
    info = new bitm_image_info
End Sub

```

### **bitm\_import\_volume\_mask\_RW.init:**

```

Sub init(input as bitm_image_info, in_map_f as folderItem, in_bitmaps_f as folderItem)
    info = input
    map_f = in_map_f
    bitmaps_f = in_bitmaps_f

    redim target_images(-1)
    redim target_masks(-1)
    if info.depth >= 1 then
        redim images(info.depth - 1)
        redim masks(info.depth - 1)
    else
        break
    end
    redim images(-1)

```

```

redim masks(-1)
change_ok = false
volume_popup.DeleteAllRows
for i as integer = 0 to info.depth - 1
    volume_popup.AddRow("Layer " + str(i+1))
    images.Append false
    masks.Append false
next
volume_popup.ListIndex = 0
change_ok = true
End Sub
bitmaps_f As folderitem

```

change\_ok As boolean

images() As boolean

info As bitm\_image\_info

map\_f As folderItem

masks() As boolean

target\_images() As picture

target\_masks() As picture

### **bitm\_import\_volume\_mask\_RW Control load\_image\_push:**

```

Sub Action()
    dim layer_index as integer = volume_popup.ListIndex
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_images(layer_index) = temp_pic
    ImageWell_image.Image = FileManip.scaleit_max(target_images(layer_index), 255)
    images(layer_index) = true
    dim load_ok as boolean = true
    for i as integer = 0 to UBound(target_images)
        if not (target_images(i) <> nil and target_masks(i) <> nil) then
            load_ok = false
            exit for i
        end
    next
end

```



```

    if load_ok then
        import_push.Enabled = true
    end
End Sub

```

### **bitm\_import\_volume\_mask\_RW Control import\_push:**

```

Sub Action()
    dim data as MemoryBlock = create_volume_bitmaps(info, target_images, target_masks)

    dim bw as BinaryStream
    dim ok as boolean = true
    if BitAnd(info.flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(ok)
    else
        bw = map_f.OpenAsBinaryFile(ok)
    end
    bw.LittleEndian = true
    bw.Position = info.offset
    bw.Write(data)
    bw.Close
    MsgBox("Bitmap Imported Successfully!")
    self.Close
End Sub

```

### **bitm\_import\_volume\_mask\_RW Control load\_mask\_push:**

```

Sub Action()
    dim layer_index as integer = volume_popup.ListIndex
    dim f as FolderItem = GetOpenFolderItem(image_filetypes.All)
    if f = nil then Return
    if not f.Exists then Return
    if f.OpenAsPicture = nil then Return
    dim temp_pic as Picture = f.OpenAsPicture
    dim temp_width as double = info.width
    dim temp_height as double = info.height
    dim width_scale as double = temp_pic.Width / temp_width
    dim height_scale as double = temp_pic.Height / temp_height
    if width_scale <> height_scale then
        errorbox("Error: image must be of " + str(info.width) + " x " + str(info.height) + " scale")
        return
    end
    target_masks(layer_index) = temp_pic
    ImageWell_image.Image = FileManip.scaleit_max(target_masks(layer_index), 255)
    masks(layer_index) = true
    dim load_ok as boolean = true
    for i as integer = 0 to UBound(target_masks)
        if not (target_images(i) <> nil and target_masks(i) <> nil) then
            load_ok = false
            exit for i
        end
    next
    if load_ok then
        import_push.Enabled = true
    end
End Sub

```

bitm\_import\_volume\_mask\_RW Control volume\_popup:

Sub Change()

if not change\_ok then return

dim layer\_index as integer = me.ListIndex

if images(layer\_index) then

ImageWell\_image.Image = FileManip.scaleit\_max(target\_images(layer\_index), 255)

else

ImageWell\_image.Image = nil

end

if masks(layer\_index) then

ImageWell\_mask.Image = FileManip.scaleit\_max(target\_masks(layer\_index), 255)

else

ImageWell\_mask.Image = nil

end

End Sub

End Class

## **Class snd\_editor\_control\_RW**

Inherits ContainerControl

### **snd\_editor\_control\_RW.Open:**

Sub Open()

#if DebugBuild

debug\_push.visible = true

debug\_push.Enabled = true

#else

debug\_push.visible = false

debug\_push.Enabled = false

#endif

End Sub

### **snd\_editor\_control\_RW.init:**

Sub init(in\_snd\_class\_RW as snd\_class\_RW)

snd\_data = in\_snd\_class\_RW

snd\_data.read

xbox\_check.Enabled = false

xbox\_check.Value = false

select case snd\_data.header.compression

case 0

//unknown

format\_text.Text = "Sound Format: Unknown"

case 1

//Xbox ADPCM

format\_text.text = "Sound Format: Xbox ADPCM"

xbox\_check.Enabled = true

case 2

//IMA ADPCM

format\_text.text = "Sound Format: IMA ADPCM"

```

case 3
    //Ogg Vorbis
    format_text.Text = "Sound Format: Ogg Vorbis"
end select

select case snd_data.header.sample_rate
case 22
    sample_rate_text.Text = "Sample Rate: 22.05 kHz"
case 44
    sample_rate_text.Text = "Sample Rate: 44.1 kHz"
case else
    sample_rate_text.Text = "Sample Rate: Unknown"
end select

if snd_data.header.stereo then
    channel_text.Text = "Channels: 2 (Stereo)"
else
    channel_text.Text = "Channels: 1 (Mono)"
end

change_ok = false
pitch_popup.DeleteAllRows
for i as integer = 1 to snd_data.header.pitch_count
    dim max_chr as integer = len(str(snd_data.header.pitch_count))
    dim temp_str as string = str(i)
    while temp_str.len < max_chr
        temp_str = "0" + temp_str
    wend
    pitch_popup.AddRow("Pitch Range " + temp_str)
next
change_ok = true
pitch_popup.ListIndex = 0
End Sub

snd_editor_control_RW.perm_update:
Private Sub perm_update(index as integer)
    if not change_ok then Return
    dim pitch_index as integer = pitch_popup.ListIndex
    if pitch_index < 0 or pitch_index > ubound(snd_data.pitch) then Return
    if index < 0 or index > UBound(snd_data.pitch(pitch_index).permutations) then Return

    dim format_str as string
    select case snd_data.pitch(pitch_index).permutations(index).compression
    case 1
        format_str = "Permutation Format: Xbox ADPCM"
    case 2
        format_str = "Permutation Format: IMA ADPCM"
    case 3
        format_str = "Permutation Format: Ogg Vorbis"
    case else
        format_str = "Permutation Format: Unknown"
    end select

    perm_format_text.Text = format_str

```

```

perm_offset_edit.Text = "0x" + hex(snd_data.pitch(pitch_index).permutations(index).offset)
perm_size_edit.Text = "0x" + hex(snd_data.pitch(pitch_index).permutations(index).size)
dim source_str as string
if snd_data.pitch(pitch_index).permutations(index).internal then
    source_str = "Source: Internal"
else
    source_str = "Source: Sounds.map"
end
End Sub

```

### **snd\_editor\_control\_RW.pitch\_update:**

```

Private Sub pitch_update(index as integer)
    change_ok = false
    if index > UBound(snd_data.pitch) or index < 0 then
        return
    end

    //permutations
    perm_list.DeleteAllRows
    for i as integer = 0 to UBound(snd_data.pitch(index).permutations)
        dim max_chr as integer = len(str(UBound(snd_data.pitch(index).permutations)+1))
        dim temp_str as string = str(i+1)
        while temp_str.len < max_chr
            temp_str = "0" + temp_str
        wend
        perm_list.AddRow("Permutation " + temp_str)
    next
    change_ok = true
    perm_list.ListIndex = 0
    perm_update(0)
End Sub
change_ok As boolean

snd_data As snd_class_RW

```

### **snd\_editor\_control\_RW Note: permutation all**

```

permutation all

dim output_f as FolderItem = SelectFolder
if output_f = nil then Return
if output_f.Exists = false then Return

for i as integer = 0 to UBound(snd_data.pitch)
    for j as integer = 0 to UBound(snd_data.pitch(i).permutations)
        dim suffix_str as string = ""
        select case snd_data.pitch(i).permutations(j).compression
            case 1
                suffix_str = ".wav"
            case 2
                suffix_str = ".wav"
            case 3
                suffix_str = ".ogg"
        end select
    next j
next i

```

```

dim write_f as FolderItem = output_f.Child(str(i) + "_" + str(j) + suffix_str)
dim bw as BinaryStream = write_f.CreateBinaryFile("")
bw.LittleEndian = true
dim br as BinaryStream
if snd_data.pitch(i).permutations(j).internal then
br = snd_data.f.OpenAsBinaryFile
else
br = snd_data.sound_f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = snd_data.pitch(i).permutations(j).offset

select case snd_data.pitch(i).permutations(j).compression
case 1, 2
//generate a RIFF file header
bw.Write("RIFF")
//precalculate the length of the entire file
dim total_file_length as integer = snd_data.pitch(i).permutations(j).size
total_file_length = total_file_length + 48 //total size of header not including RIFF header
bw.writeint32(total_file_length)

//write the wave file header
bw.write("WAVE")
//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.pitch(i).permutations(j).compression
case 1
//format_code = &h69 <- xbox adpcm codec however seems to play exactly the same as IMA
format_code = &h11
case 2
format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
sample_rate = 22050
//sample_rate = 22000
case 44
sample_rate = 44100
//sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)

```

```

bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(snd_data.pitch(i).permutations(j).size)
end select

for k as integer = 1 to snd_data.pitch(i).permutations(j).size
bw.WriteByte(br.ReadByte)
next
br.Close
bw.Close
next
next

```

```
MsgBox("complete")
```

**snd\_editor\_control\_RW Note: tracks all**  
tracks all

```

dim output_f as FolderItem = SelectFolder
if output_f = nil then Return
if output_f.Exists = false then Return

dim suffix_str as string = ""
select case snd_data.header.compression
case 1, 2
suffix_str = ".wav"
case 3
suffix_str = ".ogg"
end select

for i as integer = 0 to UBound(snd_data.pitch)
dim current_index as integer = 0
dim previous_index as integer = 0
while previous_index < snd_data.pitch(i).permutations.ubound
dim write_f as FolderItem = output_f.Child(str(previous_index) + "_" + str(i) + suffix_str)
dim bw as BinaryStream = write_f.CreateBinaryFile("")
bw.LittleEndian = true
dim start as Boolean = true

dim total_size as integer = 0
select case snd_data.header.compression
case 1,2
//generate a wave header that'll be edited in the end
//generate a RIFF file header
bw.Write("RIFF")
//precalculate the length of the entire file
dim total_file_length as integer = 0
bw.writeint32(total_file_length)

```

```

//write the wave file header
bw.write("WAVE")
//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.header.compression
case 1
//format_code = &h69 <- xbox adpcm codec however seems to play exactly the same as IMA
format_code = &h11
case 2
format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
sample_rate = 22050
//sample_rate = 22000
case 44
sample_rate = 44100
//sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.write("data")
bw.WriteUInt32(0)
end select

while current_index <> -1
dim br as BinaryStream
if snd_data.pitch(i).permutations(current_index).internal then
br = snd_data.f.OpenAsBinaryFile
else
br = snd_data.sound_f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = snd_data.pitch(i).permutations(current_index).offset

total_size = total_size + snd_data.pitch(i).permutations(current_index).size
for j as integer = 1 to snd_data.pitch(i).permutations(current_index).size
bw.WriteByte(br.ReadByte)

```

```

next
br.Close
previous_index = current_index
current_index = snd_data.pitch(i).permutations(current_index).next_perm
start = false
wend
bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
bw.Close
current_index = previous_index + 1
wend
next

```

MsgBox("complete")

**snd\_editor\_control\_RW Control debug\_push:**

```

Sub Action()
    break
End Sub

```

**snd\_editor\_control\_RW Control pitch\_popup:**

```

Sub Change()
    if change_ok then
        pitch_update(me.ListIndex)
    end
End Sub

```

**snd\_editor\_control\_RW Control perm\_list:**

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end

    if ok then
        if change_ok then perm_update(row)
    end
End Function

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean



```
if change_ok then perm_update(row)
```

End Function

## **snd\_editor\_control\_RW Control perm\_export\_push:**

Sub Action()

```
dim index as integer = pitch_popup.ListIndex
if index <= -1 OR index > UBound(snd_data.pitch) then Return
dim i as integer = perm_list.ListIndex
if i <= -1 OR i > UBound(snd_data.pitch(index).permutations) then Return
dim str_ext as string = ""
Dim soundtype as New FileType
select case snd_data.header.compression
case 1,2
    str_ext = ".wav"
    soundtype.Name = "audio/wav"
    soundtype.MacType = "RIFF"
    soundtype.Extensions = "wav"
case 3
    str_ext = ".ogg"
    soundtype.Name = "audio/ogg"
    soundtype.MacType = "OggS"
    soundtype.Extensions = "ogg"
end select
dim num_str as string = str(i+1)
dim file_str as string = "Permutation " + num_str + str_ext
dim f as FolderItem = GetSaveFolderItem(soundtype, file_str)
if f = nil then Return
dim bw as BinaryStream = f.CreateBinaryFile(f.name)
bw.LittleEndian = true

select case snd_data.header.compression
case 1,2
    //generate a wave header that'll be edited in the end
    //generate a RIFF file header
    bw.Write("RIFF")
    //precalculate the length of the entire file
    dim total_file_length as integer = 0
    bw.writeint32(total_file_length)

    //write the wave file header
    bw.write("WAVE")
    //write the format section
    bw.write("fmt" + chr(&h20))
    bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
    dim format_code as integer = 0
    select case snd_data.header.compression
    case 1
        if xbox_check.value then
            format_code = &h69 //export using the xbox adpcm value
        else
            format_code = &h11 //export using IMA for mac users
        end
    case 2
```

```

    format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
    sample_rate = 22050
    //sample_rate = 22000
case 44
    sample_rate = 44100
    //sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
dim br as BinaryStream
dim offset as integer
dim size as integer
if snd_data.pitch(index).permutations(i).internal then
    br = snd_data.f.OpenAsBinaryFile
else
    br = snd_data.sound_f.OpenAsBinaryFile
end
offset = snd_data.pitch(index).permutations(i).offset
size = snd_data.pitch(index).permutations(i).size
total_size = total_size + size
br.LittleEndian = true
br.Position = offset
for k as integer = 1 to size
    bw.WriteByte(br.ReadByte)
next
br.close

bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
case 3
    //ogg files

```

```

    dim br as BinaryStream
    dim offset as integer
    dim size as integer
    if snd_data.pitch(index).permutations(i).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(i).offset
    size = snd_data.pitch(index).permutations(i).size
    br.LittleEndian = true
    br.Position = offset
    for k as integer = 1 to size
        bw.WriteByte(br.ReadByte)
    next
    br.close

end select

bw.close
MsgBox("Permutation Exported Successfully!")
End Sub

snd_editor_control_RW Control perm_all_push:

Sub Action()
    dim f as FolderItem = SelectFolder
    if f = nil then Return
    if f.Exists = false then return
    dim index as integer = pitch_popup.ListIndex
    if index <= -1 OR index > UBound(snd_data.pitch) then Return

    dim str_ext as string = ""
    select case snd_data.header.compression
    case 1,2
        str_ext = ".wav"
    case 3
        str_ext = ".ogg"
    end select
    for i as integer = 0 to snd_data.pitch(index).permutations.ubound
        dim max_chr as integer = len(str(snd_data.pitch(index).permutations.ubound))
        dim num_str as string = str(i+1)
        while num_str.len < max_chr
            num_str = "0" + num_str
        wend
        dim output_f as FolderItem = f.Child("Permutation " + num_str + str_ext)
        dim bw as BinaryStream = output_f.CreateBinaryFile("")
        bw.LittleEndian = true

        select case snd_data.header.compression
        case 1,2
            //generate a wave header that'll be edited in the end
            //generate a RIFF file header
            bw.Write("RIFF")
            //precalculate the length of the entire file

```

```

dim total_file_length as integer = 0
bw.writeint32(total_file_length)

//write the wave file header
bw.write("WAVE")
//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.header.compression
case 1
    if xbox_check.value then
        format_code = &h69 //export using the xbox adpcm value
    else
        format_code = &h11 //export using IMA for mac users
    end
case 2
    format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
    sample_rate = 22050
    //sample_rate = 22000
case 44
    sample_rate = 44100
    //sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
dim br as BinaryStream
dim offset as integer
dim size as integer
if snd_data.pitch(index).permutations(i).internal then
    br = snd_data.f.OpenAsBinaryFile
else
    br = snd_data.sound_f.OpenAsBinaryFile

```

```

end
offset = snd_data.pitch(index).permutations(i).offset
size = snd_data.pitch(index).permutations(i).size
total_size = total_size + size
br.LittleEndian = true
br.Position = offset
for k as integer = 1 to size
    bw.WriteByte(br.ReadByte)
next
br.close

bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
case 3
    //ogg files
    dim br as BinaryStream
    dim offset as integer
    dim size as integer
    if snd_data.pitch(index).permutations(i).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(i).offset
    size = snd_data.pitch(index).permutations(i).size
    br.LittleEndian = true
    br.Position = offset
    for k as integer = 1 to size
        bw.WriteByte(br.ReadByte)
    next
    br.close

end select

bw.close
next

MsgBox("Permutations Exported Successfully!")
End Sub
End Class

```

## **Class snd\_class\_RW**

### **snd\_class\_RW.Constructor:**

```

Sub Constructor(in_sound_f as folderItem, in_f as folderitem, in_offset as integer, in_magic as integer)
    sound_f = in_sound_f
    f = in_f
    offset = in_offset
    magic = in_magic
End Sub

```

### **snd\_class\_RW.read:**

```
Sub read()  
    dim br as BinaryStream = f.OpenAsBinaryFile  
    br.LittleEndian = true  
    br.Position = offset  
    header = new snd_header_RW  
    header.read(br, magic)  
  
    br.position = header.pitch_offset  
    redim pitch(-1)  
    for i as integer = 1 to header.pitch_count  
        dim temp_pitch as new snd_pitch_RW  
        temp_pitch.read(br, magic)  
        pitch.Append temp_pitch  
    next  
End Sub
```

### **snd\_class\_RW.update\_offset:**

```
Function update_offset(pitch_ind as integer, perm_ind as integer, new_offset as integer, internal as boolean) As  
boolean  
    if pitch_ind < 0 OR pitch_ind > UBound(pitch) then Return False  
    if perm_ind < 0 or perm_ind > UBound(pitch(pitch_ind).permutations) then Return False  
  
    pitch(pitch_ind).permutations(perm_ind).offset = new_offset  
    pitch(pitch_ind).permutations(perm_ind).internal = internal  
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    pitch(pitch_ind).permutations(perm_ind).write(bw)  
  
    Return True  
End Function
```

### **snd\_class\_RW.update\_offset:**

```
Function update_offset(byref bw as binarystream, pitch_ind as integer, perm_ind as integer, new_offset as integer,  
internal as boolean) As boolean  
    if pitch_ind < 0 OR pitch_ind > UBound(pitch) then Return False  
    if perm_ind < 0 or perm_ind > UBound(pitch(pitch_ind).permutations) then Return False  
  
    pitch(pitch_ind).permutations(perm_ind).offset = new_offset  
    pitch(pitch_ind).permutations(perm_ind).internal = internal  
    bw.LittleEndian = true  
    pitch(pitch_ind).permutations(perm_ind).write(bw)  
  
    Return True  
End Function  
f As folderItem
```

header As snd\_header\_RW

magic As Integer

offset As Integer

pitch(-1) As snd\_pitch\_RW

sound\_f As folderItem

End Class

## **Class snd\_header\_RW**

### **snd\_header\_RW.read:**

Sub read(byref br as binaryStream, magic as integer)

```
    br.littleendian = true
    dim pos as integer = br.Position
    dim temp_int as integer = br.ReadUInt32
    if Bitwise.BitAnd(temp_int, 1) = 1 then
        adpcm_blocksize = true
    else
        adpcm_blocksize = false
    end
    if Bitwise.BitAnd(temp_int, 2) = 2 then
        split_into_permutations = true
    else
        split_into_permutations = false
    end
```

```
    br.Position = pos + &h6
    temp_int = br.ReadUInt16
    select case temp_int
    case 0
        sample_rate = 22
    case 1
        sample_rate = 44
    case else
        sample_rate = 0
    end select
```

```
    br.Position = pos + &h6c
    temp_int = br.ReadUInt16
    select case temp_int
    case 1
        stereo = true
    case else
        stereo = false
    end select
```

```
    br.Position = pos + &h6e
    temp_int = br.ReadUInt16
    compression = temp_int
```

```
    br.Position = pos + &h98
    pitch_count = br.ReadUInt32
    pitch_offset = br.ReadUInt32 - magic
```

```

    br.Position = pos + 164
End Sub
adpcm_blocksize As boolean

```

### **snd\_header\_RW.compression:**

```

compression As Integer
//0 = none
//1 = xbox adpcm
//2 = IMA adpcm
//3 = ogg vorbis
pitch_count As Integer

pitch_offset As Integer

sample_rate As Integer

split_into_permutations As boolean

stereo As boolean

End Class

```

## **Class snd\_pitch\_RW**

### **snd\_pitch\_RW.read:**

```

Sub read(byref br as binaryStream, magic as integer)
    br.LittleEndian = true
    dim pos as integer = br.Position
    br.Position = pos + &h3c
    perm_count = br.ReadUInt32
    perm_offset = br.ReadUInt32 - magic

    br.Position = pos + &h2c
    track_count = br.ReadUInt16

    br.position = perm_offset
    redim permutations(-1)
    for i as integer = 1 to perm_count
        dim temp_perm as new snd_perm
        temp_perm.read(br, magic)
        permutations.Append temp_perm
    next

    br.Position = pos + 72

    //problems related to track stuff
    'redim tracks(-1)
    'dim current_index as integer = 0
    'dim previous_index as integer = 0
    'dim index_ar(-1) as integer
    'for i as integer = 0 to UBound(permutations)

```



```

'index_ar.append i
'next
'while previous_index <= permutations.ubound AND current_index <= permutations.Ubound AND current_index
<> -1
'dim start as Boolean = true
'dim temp_track as new snd_track
'while current_index <> -1
'temp_track.perm_list.Append current_index
'index_ar(current_index) = -1
'previous_index = current_index
'if permutations(current_index).next_perm = current_index then
'current_index = -1
'else
'current_index = permutations(current_index).next_perm
'end
'start = false
'wend
'tracks.Append temp_track
'for i as integer = 0 to UBound(index_ar)
'if index_ar(i) <> -1 then
'current_index = index_ar(i)
'exit for i
'end
'next
'wend
'

'if UBound(tracks) +1 <> track_count then
'dim temp as integer //for break point
'end
'for i as integer = 0 to UBound(index_ar)
'if index_ar(i) <> -1 then
'dim temp2 as integer
'end
'next
End Sub
permutations(-1) As snd_perm

perm_count As Integer

perm_offset As Integer

tracks(-1) As snd_track

track_count As Integer

End Class

```

## **Class mod2\_header**

### **mod2\_header.read:**

```

Sub read(byref br as binaryStream, magic as integer)
    br.LittleEndian = true

```

```

dim pos as integer = br.Position

br.Position = pos + &hd0
geometries_count = br.ReadUInt32
geometries_offset = br.ReadUInt32 - magic

//need to define headersize
//br.Position = pos + headersize
End Sub
geometries_count As Integer

geometries_offset As Integer

End Class

```

## **Class mod2\_class**

### **mod2\_class.Constructor:**

```

Sub Constructor(in_f as folderitem, in_offset as integer, in_magic as integer)
    f = in_f
    offset = in_offset
    magic = in_magic
End Sub

```

### **mod2\_class.read:**

```

Sub read()
    dim br as BinaryStream = f.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    header = new mod2_header
    header.read(br, magic)

    br.Position = header.geometries_offset
    redim geometries(-1)
    for i as integer = 1 to header.geometries_count
        dim temp_geometry as new mod2_geometry
        temp_geometry.read(br, magic)
        geometries.Append temp_geometry
    next
End Sub

```

### **mod2\_class.update\_offset:**

```

Function update_offset(geometry_ind as integer, part_ind as integer, new_offset as integer, type as integer) As
boolean
    if geometry_ind < 0 OR geometry_ind > UBound(geometries) then Return False
    if part_ind < 0 OR part_ind > UBound(geometries(geometry_ind).parts) then Return False

    select case type
    case 1
        //vert
        geometries(geometry_ind).parts(part_ind).vert_offset = new_offset
    case 2

```

```

    //ind
    geometries(geometry_ind).parts(part_ind).ind_offset1 = new_offset
    geometries(geometry_ind).parts(part_ind).ind_offset2 = new_offset
end select
dim bw as BinaryStream
bw = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
geometries(geometry_ind).parts(part_ind).write(bw)

Return True
End Function

```

### **mod2\_class.update\_offset:**

Function update\_offset(byref bw as binarystream, geometry\_ind as integer, part\_ind as integer, new\_offset as integer, type as integer) As boolean

```

if geometry_ind < 0 OR geometry_ind > UBound(geometries) then Return False
if part_ind < 0 OR part_ind > UBound(geometries(geometry_ind).parts) then Return False

```

```

select case type
case 1
    //vert
    geometries(geometry_ind).parts(part_ind).vert_offset = new_offset
case 2
    //ind
    geometries(geometry_ind).parts(part_ind).ind_offset1 = new_offset
    geometries(geometry_ind).parts(part_ind).ind_offset2 = new_offset
end select
bw.LittleEndian = true
geometries(geometry_ind).parts(part_ind).write(bw)

```

```

Return True
End Function
f As folderItem

```

geometries(-1) As mod2\_geometry

header As mod2\_header

magic As Integer

offset As Integer

### **mod2\_class Note: Ind**

Ind  
Inds are at index.vert\_offset + index.ind\_offset + mod2.ind\_offset

The ind section in raw is unusual to say the least

the first ind consists of the first three shorts  
the second ind consists of the 2nd short, 3rd short and 4th short  
and so forth  
so in total there are Xinds \* 2 bytes + 4 bytes worth of ind data

mod2\_class Note: Vert

Vert

Verts are at index.vert\_offset + mod2.vert\_offset

The vert section in raw is 0x44 bytes long and of this form:

float coord[3];

float Normal[3];

float Binormal[3];

float Tangent[3];

float u;

float v;

int bone\_ref; //reference to "bone" aka rigging

float unk1; //always 1

float unk2: //always 0

End Class

## **Class mod2\_geometry**

### **mod2\_geometry.read:**

Sub read(byref br as binaryStream, magic as integer)

br.LittleEndian = true

pos = br.Position

br.Position = pos + &h24

parts\_count = br.ReadUInt32

parts\_offset = br.ReadUInt32 - magic

br.Position = parts\_offset

redim parts(-1)

for i as integer = 1 to parts\_count

dim temp\_part as new mod2\_part

temp\_part.read(br)

parts.Append temp\_part

next

br.Position = pos + &h30

End Sub

parts(-1) As mod2\_part

parts\_count As Integer

parts\_offset As Integer

pos As Integer

End Class

## **Class mod2\_part**

### **mod2\_part.read:**

```

Sub read(byref br as binaryStream)
    br.LittleEndian = true
    pos = br.Position

    br.Position = pos + &h48
    ind_count = br.ReadUInt32
    ind_offset1 = br.ReadUInt32
    ind_offset2 = br.ReadUInt32

    br.Position = pos + &h58
    vert_count = br.ReadUInt32

    br.Position = pos + &h64
    vert_offset = br.ReadUInt32

    br.Position = pos + &h84
End Sub

```

### **mod2\_part.write:**

```

Sub write(byref bw as binaryStream)
    bw.LittleEndian = true
    bw.Position = pos

    bw.Position = pos + &h48
    bw.WriteUInt32(ind_count)
    bw.WriteUInt32(ind_offset1)
    bw.WriteUInt32(ind_offset2)

    bw.Position = pos + &h58
    bw.WriteUInt32(vert_count)

    bw.Position = pos + &h64
    bw.WriteUInt32(vert_offset)

    bw.Position = pos + &h84
End Sub

```

ind\_count As Integer

ind\_offset1 As Integer

ind\_offset2 As Integer

pos As Integer

vert\_count As Integer

vert\_offset As Integer

End Class

## **Class snd\_editor\_control\_RW1**

Inherits ContainerControl

### **snd\_editor\_control\_RW1.Open:**

```
Sub Open()  
    #if DebugBuild  
        debug_push.visible = true  
        debug_push.Enabled = true  
    #else  
        debug_push.visible = false  
        debug_push.Enabled = false  
    #endif  
End Sub
```

### **snd\_editor\_control\_RW1.init:**

```
Sub init(in_snd_class_RW as snd_class_RW)  
    snd_data = in_snd_class_RW  
    snd_data.read  
  
    xbox_check.Enabled = false  
    xbox_check.Value = false  
    select case snd_data.header.compression  
    case 0  
        //unknown  
        format_text.Text = "Sound Format: Unknown"  
    case 1  
        //Xbox ADPCM  
        format_text.text = "Sound Format: Xbox ADPCM"  
        xbox_check.Enabled = true  
    case 2  
        //IMA ADPCM  
        format_text.text = "Sound Format: IMA ADPCM"  
    case 3  
        //Ogg Vorbis  
        format_text.Text = "Sound Format: Ogg Vorbis"  
    end select  
  
    select case snd_data.header.sample_rate  
    case 22  
        sample_rate_text.Text = "Sample Rate: 22.05 kHz"  
    case 44  
        sample_rate_text.Text = "Sample Rate: 44.1 kHz"  
    case else  
        sample_rate_text.Text = "Sample Rate: Unknown"  
    end select  
  
    if snd_data.header.stereo then  
        channel_text.Text = "Channels: 2 (Stereo)"  
    else  
        channel_text.Text = "Channels: 1 (Mono)"  
    end  
  
    change_ok = false  
    pitch_popup.DeleteAllRows  
    for i as integer = 1 to snd_data.header.pitch_count  
        dim max_chr as integer = len(str(snd_data.header.pitch_count))
```

```

        dim temp_str as string = str(i)
        while temp_str.len < max_chr
            temp_str = "0" + temp_str
        wend
        pitch_popup.AddRow("Pitch Range " + temp_str)
    next
    change_ok = true
    pitch_popup.ListIndex = 0
End Sub

```

### **snd\_editor\_control\_RW1.perm\_update:**

```

Private Sub perm_update(index as integer)
    if not change_ok then Return
    dim pitch_index as integer = pitch_popup.ListIndex
    if pitch_index < 0 or pitch_index > ubound(snd_data.pitch) then Return
    if index < 0 or index > UBound(snd_data.pitch(pitch_index).permutations) then Return

    dim format_str as string
    select case snd_data.pitch(pitch_index).permutations(index).compression
    case 1
        format_str = "Permutation Format: Xbox ADPCM"
    case 2
        format_str = "Permutation Format: IMA ADPCM"
    case 3
        format_str = "Permutation Format: Ogg Vorbis"
    case else
        format_str = "Permutation Format: Unknown"
    end select

    perm_format_text.Text = format_str
    perm_offset_edit.Text = "0x" + hex(snd_data.pitch(pitch_index).permutations(index).offset)
    perm_size_edit.Text = "0x" + hex(snd_data.pitch(pitch_index).permutations(index).size)
    dim source_str as string
    if snd_data.pitch(pitch_index).permutations(index).internal then
        source_str = "Source: Internal"
    else
        source_str = "Source: Sounds.map"
    end
End Sub

```

### **snd\_editor\_control\_RW1.pitch\_update:**

```

Private Sub pitch_update(index as integer)
    change_ok = false
    if index > UBound(snd_data.pitch) or index < 0 then
        return
    end

    //tracks
    track_list.DeleteAllRows
    for i as integer = 1 to snd_data.pitch(index).track_count
        dim max_chr as integer = len(str(snd_data.pitch(index).track_count))
        dim temp_str as string = str(i)
        while temp_str.len < max_chr
            temp_str = "0" + temp_str
        wend
    next
End Sub

```

```

        wend
        track_list.AddRow("Track " + temp_str)
    next
    change_ok = true
    track_list.ListIndex = 0
    track_update(0)
    change_ok = false

    //permutations
    perm_list.DeleteAllRows
    for i as integer = 0 to UBound(snd_data.pitch(index).permutations)
        dim max_chr as integer = len(str(UBound(snd_data.pitch(index).permutations)+1))
        dim temp_str as string = str(i+1)
        while temp_str.len < max_chr
            temp_str = "0" + temp_str
        wend
        perm_list.AddRow("Permutation " + temp_str)
    next
    change_ok = true
    perm_list.ListIndex = 0
    perm_update(0)
End Sub

```

### **snd\_editor\_control\_RW1.track\_update:**

```

Private Sub track_update(index as integer)
    if not change_ok then Return
    dim pitch_index as integer = pitch_popup.ListIndex
    if pitch_index < 0 or pitch_index > ubound(snd_data.pitch) then Return
    if index < 0 or index > UBound(snd_data.pitch(pitch_index).tracks) then Return

    dim temp_str as string = "Included Permutations: "
    dim start as boolean = true
    for i as integer = 0 to UBound(snd_data.pitch(pitch_index).tracks(index).perm_list)
        dim max_chr as integer = len(str(UBound(snd_data.pitch(pitch_index).tracks(index).perm_list)))
        dim num_str as string = str(snd_data.pitch(pitch_index).tracks(index).perm_list(i) + 1)
        while num_str.len < max_chr
            num_str = "0" + num_str
        wend
        if not start then num_str = ", " + num_str
        start = false
        temp_str = temp_str + num_str
    next
    track_text.Text = temp_str
End Sub
change_ok As boolean

```

snd\_data As snd\_class\_RW

### **snd\_editor\_control\_RW1 Note: permutation all** permutation all

```

dim output_f as FolderItem = SelectFolder
if output_f = nil then Return

```



```

if output_f.Exists = false then Return

for i as integer = 0 to UBound(snd_data.pitch)
for j as integer = 0 to UBound(snd_data.pitch(i).permutations)
dim suffix_str as string = ""
select case snd_data.pitch(i).permutations(j).compression
case 1
suffix_str = ".wav"
case 2
suffix_str = ".wav"
case 3
suffix_str = ".ogg"
end select

dim write_f as FolderItem = output_f.Child(str(i) + "_" + str(j) + suffix_str)
dim bw as BinaryStream = write_f.CreateBinaryFile("")
bw.LittleEndian = true
dim br as BinaryStream
if snd_data.pitch(i).permutations(j).internal then
br = snd_data.f.OpenAsBinaryFile
else
br = snd_data.sound_f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = snd_data.pitch(i).permutations(j).offset

select case snd_data.pitch(i).permutations(j).compression
case 1, 2
//generate a RIFF file header
bw.Write("RIFF")
//precalculate the length of the entire file
dim total_file_length as integer = snd_data.pitch(i).permutations(j).size
total_file_length = total_file_length + 48 //total size of header not including RIFF header
bw.writeint32(total_file_length)

//write the wave file header
bw.write("WAVE")
//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.pitch(i).permutations(j).compression
case 1
//format_code = &h69 <- xbox adpcm codec however seems to play exactly the same as IMA
format_code = &h11
case 2
format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate

```

```

case 22
sample_rate = 22050
//sample_rate = 22000
case 44
sample_rate = 44100
//sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(snd_data.pitch(i).permutations(j).size)
end select

for k as integer = 1 to snd_data.pitch(i).permutations(j).size
bw.WriteByte(br.ReadByte)
next
br.Close
bw.Close
next
next

MsgBox("complete")

```

## **snd\_editor\_control\_RW1 Note: tracks all**

tracks all

```

dim output_f as FolderItem = SelectFolder
if output_f = nil then Return
if output_f.Exists = false then Return

dim suffix_str as string = ""
select case snd_data.header.compression
case 1, 2
suffix_str = ".wav"
case 3
suffix_str = ".ogg"
end select

for i as integer = 0 to UBound(snd_data.pitch)
dim current_index as integer = 0
dim previous_index as integer = 0
while previous_index < snd_data.pitch(i).permutations.ubound
dim write_f as FolderItem = output_f.Child(str(previous_index) + "_" + str(i) + suffix_str)
dim bw as BinaryStream = write_f.CreateBinaryFile("")

```

```

bw.LittleEndian = true
dim start as Boolean = true

dim total_size as integer = 0
select case snd_data.header.compression
case 1,2
//generate a wave header that'll be edited in the end
//generate a RIFF file header
bw.Write("RIFF")
//precalculate the length of the entire file
dim total_file_length as integer = 0
bw.writeint32(total_file_length)

//write the wave file header
bw.write("WAVE")
//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.header.compression
case 1
//format_code = &h69 <- xbox adpcm codec however seems to play exactly the same as IMA
format_code = &h11
case 2
format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
sample_rate = 22050
//sample_rate = 22000
case 44
sample_rate = 44100
//sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(0)
end select

```

```

while current_index <> -1
dim br as BinaryStream
if snd_data.pitch(i).permutations(current_index).internal then
br = snd_data.f.OpenAsBinaryFile
else
br = snd_data.sound_f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = snd_data.pitch(i).permutations(current_index).offset

total_size = total_size + snd_data.pitch(i).permutations(current_index).size
for j as integer = 1 to snd_data.pitch(i).permutations(current_index).size
bw.WriteByte(br.ReadByte)
next
br.Close
previous_index = current_index
current_index = snd_data.pitch(i).permutations(current_index).next_perm
start = false
wend
bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
bw.Close
current_index = previous_index + 1
wend
next

```

MsgBox("complete")

### **snd\_editor\_control\_RW1 Control debug\_push:**

```

Sub Action()
    break
End Sub

```

### **snd\_editor\_control\_RW1 Control pitch\_popup:**

```

Sub Change()
    if change_ok then
        pitch_update(me.ListIndex)
    end
End Sub

```

### **snd\_editor\_control\_RW1 Control track\_list:**

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    if change_ok then track_update(row)
End Function

```

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31

```

```

    if me.ListIndex + 1 < me.listcount then
        row = me.ListIndex + 1
        ok = true
    end
case 30
    if me.ListIndex - 1 > -1 then
        row = me.ListIndex - 1
        ok = true
    end
end

if ok then
    if change_ok then track_update(row)
end
End Function

```

### **snd\_editor\_control\_RW1 Control track\_export\_push:**

```

Sub Action()
    dim index as integer = pitch_popup.ListIndex
    if index <= -1 OR index > UBound(snd_data.pitch) then Return
    dim i as integer = track_list.ListIndex
    if i <= -1 OR i > UBound(snd_data.pitch(index).tracks) then Return
    dim str_ext as string = ""
    Dim soundtype as New FileType
    select case snd_data.header.compression
    case 1,2
        str_ext = ".wav"
        soundtype.Name = "audio/wav"
        soundtype.MacType = "RIFF"
        soundtype.Extensions = "wav"
    case 3
        str_ext = ".ogg"
        soundtype.Name = "audio/ogg"
        soundtype.MacType = "OggS"
        soundtype.Extensions = "ogg"
    end select
    dim num_str as string = str(i+1)
    dim file_str as string = "Track " + num_str + str_ext
    dim f as FolderItem = GetSaveFolderItem(soundtype, file_str)
    if f = nil then Return
    dim bw as BinaryStream = f.CreateBinaryFile(f.name)
    bw.LittleEndian = true

    select case snd_data.header.compression
    case 1,2
        //generate a wave header that'll be edited in the end
        //generate a RIFF file header
        bw.Write("RIFF")
        //precalculate the length of the entire file
        dim total_file_length as integer = 0
        bw.writeint32(total_file_length)

        //write the wave file header
        bw.write("WAVE")
    end select
end Sub

```

```

//write the format section
bw.write("fmt" + chr(&h20))
bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
dim format_code as integer = 0
select case snd_data.header.compression
case 1
    if xbox_check.value then
        format_code = &h69 //export using the xbox adpcm value
    else
        format_code = &h11 //export using IMA for mac users
    end
case 2
    format_code = &h11
end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
    sample_rate = 22050
    //sample_rate = 22000
case 44
    sample_rate = 44100
    //sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
for j as integer = 0 to UBound(snd_data.pitch(index).tracks(i).perm_list)
    dim br as BinaryStream
    dim offset as integer
    dim size as integer
    dim perm_index as integer = snd_data.pitch(index).tracks(i).perm_list(j)
    if snd_data.pitch(index).permutations(perm_index).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(perm_index).offset
    size = snd_data.pitch(index).permutations(perm_index).size

```

```

        total_size = total_size + size
        br.LittleEndian = true
        br.Position = offset
        for k as integer = 1 to size
            bw.WriteByte(br.ReadByte)
        next
        br.close
    next

    bw.Position = 4
    bw.WriteUInt32(total_size + 48)
    bw.Position = 44
    bw.WriteUInt32(total_size)
case 3
    //ogg files
    for j as integer = 0 to UBound(snd_data.pitch(index).tracks(i).perm_list)
        dim br as BinaryStream
        dim offset as integer
        dim size as integer
        dim perm_index as integer = snd_data.pitch(index).tracks(i).perm_list(j)
        if snd_data.pitch(index).permutations(perm_index).internal then
            br = snd_data.f.OpenAsBinaryFile
        else
            br = snd_data.sound_f.OpenAsBinaryFile
        end
        offset = snd_data.pitch(index).permutations(perm_index).offset
        size = snd_data.pitch(index).permutations(perm_index).size
        br.LittleEndian = true
        br.Position = offset
        for k as integer = 1 to size
            bw.WriteByte(br.ReadByte)
        next
        br.close
    next

end select

bw.close
MsgBox("Track Exported Successfully!")
End Sub

```

### **snd\_editor\_control\_RW1 Control track\_all\_push:**

```

Sub Action()
    dim f as FolderItem = SelectFolder
    if f = nil then Return
    if f.Exists = false then return
    dim index as integer = pitch_popup.ListIndex
    if index <= -1 OR index > UBound(snd_data.pitch) then Return

    dim str_ext as string = ""
    select case snd_data.header.compression
    case 1,2
        str_ext = ".wav"
    case 3

```

```

    str_ext = ".ogg"
end select
for i as integer = 0 to snd_data.pitch(index).tracks.ubound
    dim max_chr as integer = len(str(snd_data.pitch(index).track_count))
    dim num_str as string = str(i+1)
    while num_str.len < max_chr
        num_str = "0" + num_str
    wend
    dim output_f as FolderItem = f.Child("Track " + num_str + str_ext)
    dim bw as BinaryStream = output_f.CreateBinaryFile("")
    bw.LittleEndian = true

    select case snd_data.header.compression
    case 1,2
        //generate a wave header that'll be edited in the end
        //generate a RIFF file header
        bw.Write("RIFF")
        //precalculate the length of the entire file
        dim total_file_length as integer = 0
        bw.writeint32(total_file_length)

        //write the wave file header
        bw.write("WAVE")
        //write the format section
        bw.write("fmt" + chr(&h20))
        bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
        dim format_code as integer = 0
        select case snd_data.header.compression
        case 1
            if xbox_check.value then
                format_code = &h69 //export using the xbox adpcm value
            else
                format_code = &h11 //export using IMA for mac users
            end
        case 2
            format_code = &h11
        end select
        bw.WriteUInt16(format_code) //write the format code
        dim num_channels as integer = 0
        if snd_data.header.stereo then num_channels = 2 else num_channels = 1
        bw.WriteUInt16(num_channels) //write how many channels there are
        dim sample_rate as integer = 0
        select case snd_data.header.sample_rate
        case 22
            sample_rate = 22050
            //sample_rate = 22000
        case 44
            sample_rate = 44100
            //sample_rate = 44000
        end select
        bw.WriteUInt32(sample_rate) //write the sample rate
        dim significant_bits_per_sample as integer = 4
        dim block_align as integer = num_channels * 36 //hmt info
        dim byte_rate as integer = num_channels * sample_rate / 2
    end select
end for

```



```

bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
for j as integer = 0 to UBound(snd_data.pitch(index).tracks(i).perm_list)
    dim br as BinaryStream
    dim offset as integer
    dim size as integer
    dim perm_index as integer = snd_data.pitch(index).tracks(i).perm_list(j)
    if snd_data.pitch(index).permutations(perm_index).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(perm_index).offset
    size = snd_data.pitch(index).permutations(perm_index).size
    total_size = total_size + size
    br.LittleEndian = true
    br.Position = offset
    for k as integer = 1 to size
        bw.WriteByte(br.ReadByte)
    next
    br.close
next

bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
case 3
//ogg files
for j as integer = 0 to UBound(snd_data.pitch(index).tracks(i).perm_list)
    dim br as BinaryStream
    dim offset as integer
    dim size as integer
    dim perm_index as integer = snd_data.pitch(index).tracks(i).perm_list(j)
    if snd_data.pitch(index).permutations(perm_index).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(perm_index).offset
    size = snd_data.pitch(index).permutations(perm_index).size
    br.LittleEndian = true
    br.Position = offset
    for k as integer = 1 to size
        bw.WriteByte(br.ReadByte)
    next
next

```

```

        next
    br.close
next

end select

bw.close
next

MsgBox("Tracks Exported Successfully!")
End Sub

snd_editor_control_RW1 Control perm_list:

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim column as integer = 0
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end
end

if ok then
    if change_ok then perm_update(row)
end
End Function

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    if change_ok then perm_update(row)

End Function

snd_editor_control_RW1 Control perm_export_push:

Sub Action()
    dim index as integer = pitch_popup.ListIndex
    if index <= -1 OR index > UBound(snd_data.pitch) then Return
    dim i as integer = perm_list.ListIndex
    if i <= -1 OR i > UBound(snd_data.pitch(index).permutations) then Return
    dim str_ext as string = ""
    Dim soundtype as New FileType
    select case snd_data.header.compression
    case 1,2
        str_ext = ".wav"
        soundtype.Name = "audio/wav"
        soundtype.MacType = "RIFF"
        soundtype.Extensions = "wav"
    
```

```

case 3
    str_ext = ".ogg"
    soundtype.Name = "audio/ogg"
    soundtype.MacType = "OggS"
    soundtype.Extensions = "ogg"
end select
dim num_str as string = str(i+1)
dim file_str as string = "Permutation " + num_str + str_ext
dim f as FolderItem = GetSaveFolderItem(soundtype, file_str)
if f = nil then Return
dim bw as BinaryStream = f.CreateBinaryFile(f.name)
bw.LittleEndian = true

select case snd_data.header.compression
case 1,2
    //generate a wave header that'll be edited in the end
    //generate a RIFF file header
    bw.Write("RIFF")
    //precalculate the length of the entire file
    dim total_file_length as integer = 0
    bw.writeint32(total_file_length)

    //write the wave file header
    bw.write("WAVE")
    //write the format section
    bw.write("fmt" + chr(&h20))
    bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
    dim format_code as integer = 0
    select case snd_data.header.compression
    case 1
        if xbox_check.value then
            format_code = &h69 //export using the xbox adpcm value
        else
            format_code = &h11 //export using IMA for mac users
        end
    case 2
        format_code = &h11
    end select
    bw.WriteUInt16(format_code) //write the format code
    dim num_channels as integer = 0
    if snd_data.header.stereo then num_channels = 2 else num_channels = 1
    bw.WriteUInt16(num_channels) //write how many channels there are
    dim sample_rate as integer = 0
    select case snd_data.header.sample_rate
    case 22
        sample_rate = 22050
        //sample_rate = 22000
    case 44
        sample_rate = 44100
        //sample_rate = 44000
    end select
    bw.WriteUInt32(sample_rate) //write the sample rate
    dim significant_bits_per_sample as integer = 4
    dim block_align as integer = num_channels * 36 //hmt info

```

```

dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
dim br as BinaryStream
dim offset as integer
dim size as integer
if snd_data.pitch(index).permutations(i).internal then
    br = snd_data.f.OpenAsBinaryFile
else
    br = snd_data.sound_f.OpenAsBinaryFile
end
offset = snd_data.pitch(index).permutations(i).offset
size = snd_data.pitch(index).permutations(i).size
total_size = total_size + size
br.LittleEndian = true
br.Position = offset
for k as integer = 1 to size
    bw.WriteByte(br.ReadByte)
next
br.close

bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
case 3
//ogg files
dim br as BinaryStream
dim offset as integer
dim size as integer
if snd_data.pitch(index).permutations(i).internal then
    br = snd_data.f.OpenAsBinaryFile
else
    br = snd_data.sound_f.OpenAsBinaryFile
end
offset = snd_data.pitch(index).permutations(i).offset
size = snd_data.pitch(index).permutations(i).size
br.LittleEndian = true
br.Position = offset
for k as integer = 1 to size
    bw.WriteByte(br.ReadByte)
next
br.close

end select

```

```

    bw.close
    MsgBox("Permutation Exported Successfully!")
End Sub

snd_editor_control_RW1 Control perm_all_push:

```

```

Sub Action()
    dim f as FolderItem = SelectFolder
    if f = nil then Return
    if f.Exists = false then return
    dim index as integer = pitch_popup.ListIndex
    if index <= -1 OR index > UBound(snd_data.pitch) then Return

    dim str_ext as string = ""
    select case snd_data.header.compression
    case 1,2
        str_ext = ".wav"
    case 3
        str_ext = ".ogg"
    end select
    for i as integer = 0 to snd_data.pitch(index).permutations.ubound
        dim max_chr as integer = len(str(snd_data.pitch(index).permutations.ubound))
        dim num_str as string = str(i+1)
        while num_str.len < max_chr
            num_str = "0" + num_str
        wend
        dim output_f as FolderItem = f.Child("Permutation " + num_str + str_ext)
        dim bw as BinaryStream = output_f.CreateBinaryFile("")
        bw.LittleEndian = true

        select case snd_data.header.compression
        case 1,2
            //generate a wave header that'll be edited in the end
            //generate a RIFF file header
            bw.Write("RIFF")
            //precalculate the length of the entire file
            dim total_file_length as integer = 0
            bw.writeint32(total_file_length)

            //write the wave file header
            bw.write("WAVE")
            //write the format section
            bw.write("fmt" + chr(&h20))
            bw.WriteUInt32(20) //always 16 bytes long + extra data. 4 bytes extra always
            dim format_code as integer = 0
            select case snd_data.header.compression
            case 1
                if xbox_check.value then
                    format_code = &h69 //export using the xbox adpcm value
                else
                    format_code = &h11 //export using IMA for mac users
                end
            case 2
                format_code = &h11
            end select
        end select
    next i
End Sub

```

```

end select
bw.WriteUInt16(format_code) //write the format code
dim num_channels as integer = 0
if snd_data.header.stereo then num_channels = 2 else num_channels = 1
bw.WriteUInt16(num_channels) //write how many channels there are
dim sample_rate as integer = 0
select case snd_data.header.sample_rate
case 22
    sample_rate = 22050
    //sample_rate = 22000
case 44
    sample_rate = 44100
    //sample_rate = 44000
end select
bw.WriteUInt32(sample_rate) //write the sample rate
dim significant_bits_per_sample as integer = 4
dim block_align as integer = num_channels * 36 //hmt info
dim byte_rate as integer = num_channels * sample_rate / 2
bw.WriteUInt32(byte_rate)
bw.WriteUInt16(block_align)
bw.WriteUInt16(significant_bits_per_sample)
bw.WriteUInt16(2) //extra parameter size
bw.WriteUInt16(64) //data related to ADPCM from HMT
//write the data section
//NOTE: hmt doesn't seem to use the fact section
bw.Write("data")
bw.WriteUInt32(0)

dim total_size as integer = 0
dim br as BinaryStream
dim offset as integer
dim size as integer
if snd_data.pitch(index).permutations(i).internal then
    br = snd_data.f.OpenAsBinaryFile
else
    br = snd_data.sound_f.OpenAsBinaryFile
end
offset = snd_data.pitch(index).permutations(i).offset
size = snd_data.pitch(index).permutations(i).size
total_size = total_size + size
br.LittleEndian = true
br.Position = offset
for k as integer = 1 to size
    bw.WriteByte(br.ReadByte)
next
br.close

bw.Position = 4
bw.WriteUInt32(total_size + 48)
bw.Position = 44
bw.WriteUInt32(total_size)
case 3
    //ogg files
    dim br as BinaryStream

```

```

    dim offset as integer
    dim size as integer
    if snd_data.pitch(index).permutations(i).internal then
        br = snd_data.f.OpenAsBinaryFile
    else
        br = snd_data.sound_f.OpenAsBinaryFile
    end
    offset = snd_data.pitch(index).permutations(i).offset
    size = snd_data.pitch(index).permutations(i).size
    br.LittleEndian = true
    br.Position = offset
    for k as integer = 1 to size
        bw.WriteByte(br.ReadByte)
    next
    br.close

end select

    bw.close
next

MsgBox("Permutations Exported Successfully!")
End Sub
End Class

```

## Class Model\_Header

### **Model\_Header.read:**

```

Sub read(byref br as binaryStream)
    br.LittleEndian = true
    dim temp_pos as integer = br.position
    dim temp as new reflexive

    zero1 = br.ReadUInt32

    unknown1a = br.ReadUInt32

    offset1 = br.ReadUInt32

    offset2 = br.ReadUInt32

    offset3 = br.ReadUInt32

    offset4 = br.ReadUInt32

    offset5 = br.ReadUInt32

    ReDim Lod_cutoffs(-1)

```

```

for i as integer = 0 to 4
    Lod_cutoffs.append br.ReadInt16
next

ReDim unknown2(4)

For x As Integer = 0 To 4

    unknown2(x) = br.ReadInt16

Next

uScale = br.ReadSingle

vScale = br.ReadSingle

ReDim unknown3(29)

For x As Integer = 1 To 29

    unknown3(x) = br.ReadUInt32

Next x

AttachmentPoints = temp.read(br)

Bones = temp.read(br)

Models = temp.read(br)

SubModels = temp.read(br)

    Shaders = temp.read(br)
End Sub
attachmentPoints As reflexive

bones As reflexive

Lod_cutoffs() As Integer

models As reflexive

offset1 As uint32

offset2 As uint32

offset3 As uint32

offset4 As uint32

offset5 As uint32

shaders As reflexive

```



submodels As reflexive

unknown1a As uint32

unknown1b() As uint32

unknown2() As short

unknown3() As uint32

uScale As Single

vScale As single

zero1 As uint32

End Class

## **Class Model\_Reflexive\_data**

### **Model\_Reflexive\_data.read:**

Sub read(byref br as binaryStream, magic as integer)

    br.LittleEndian = true

    dim position as integer = br.position

    dim temp as new reflexive

    Dim temp\_str As string = ""

    Dim inputint as int8

    inputint = br.readint8

    br.position = br.position - 1

    while (not(inputint = 0))

        temp\_str = temp\_str + br.read(1)

        inputint = br.readint8

        br.position = br.position - 1

    wend

    name = temp\_str

    redim zeros(-1)

    br.position = position + 32

    for i as integer = 0 to 7

        zeros.append br.ReadUInt32

    next

    chunk1 = temp.read(br)

    redim links(-1)

    for i as integer = 0 to chunk1.count-1

        dim temp\_link as new Model\_Reflexive\_link

        temp\_link.read(br, chunk1.offset + (i\*88) - magic, name)

        links.Append temp\_link

    next

```
End Sub
chunk1 As reflexive
```

```
links() As Model_Reflexive_Link
```

```
name As string
```

```
zeros() As uint32
```

```
End Class
```

## **Class Model Reflexive link**

### **Model\_Reflexive\_link.read:**

```
Sub read(byref br as binarystream, offset as integer, in_name as string)
```

```
    br.position = offset
```

```
    br.littleEndian = true
```

```
    Dim temp_str As string = ""
```

```
    Dim inputint as int8
```

```
    inputint = br.readint8
```

```
    br.position = br.position - 1
```

```
    while (not(inputint = 0))
```

```
        temp_str = temp_str + br.read(1)
```

```
        inputint = br.readint8
```

```
        br.position = br.position - 1
```

```
    wend
```

```
    name = in_name + temp_str
```

```
    br.position = offset + &h20
```

```
    unknown = br.readint32
```

```
    br.position = offset + &h40
```

```
    super_low = br.readint16
```

```
    br.position = offset + &h42
```

```
    low = br.readint16
```

```
    br.position = offset + &h44
```

```
    medium = br.readint16
```

```
    br.position = offset + &h46
```

```
    high = br.readint16
```

```
    br.position = offset + &h48
```

```
    super_high = br.readint16
```

```
End Sub
```

```
high As Integer
```

```
low As Integer
```

```
medium As Integer
```

name As string

super\_high As Integer

super\_low As Integer

unknown As Integer

About

This contains all the indices of the submodels that involve the main model (possibly with relation to the LoD order)  
End Class

## **Class SubModel\_Header**

### **SubModel\_Header.read:**

```
Sub read(byref br as binaryStream)
    br.LittleEndian = true
    dim temp as integer = br.position
    count = br.ReadInt32
    offset = br.ReadUInt32
    redim zeros(9)
    for x as integer = 0 to 9
        zeros(x) = br.ReadInt32
    next
End Sub
count As Integer
```

offset As uint32

zeros() As Integer

End Class

## **Class Model\_TriangleStrip\_Header**

### **Model\_TriangleStrip\_Header.read:**

```
Sub read(byref br as binaryStream)
    br.LittleEndian = true

    Dim x As Integer

    ReDim unknown(6)

    ReDim zeros(9)

    ReDim unknown2(2)

    ReDim zeros2(7)
```

```

StartingOffset = br.Position

zero = br.ReadInt32()

number = br.ReadInt16()

effeff = br.ReadInt16()

For x = 0 To 5

    unknown(x) = br.ReadInt32()

Next

For x = 0 To 8

    zeros(x) = br.ReadInt32()

Next

one = br.ReadInt32()

index_count = br.ReadInt32() //+ 2

index_offset = br.ReadInt32()

unknown1_offset = br.ReadInt32()

unknown1_count = br.ReadInt32()

vertex_count = br.ReadInt32()

unknown2(1) = br.ReadInt32()

unknown2(2) = br.ReadInt32()

vertex_offset = br.ReadInt32()

For x = 0 To 6 ' PC ONLY

    zeros2(x) = br.ReadInt32()

Next
End Sub
effeff As short

index_count As Integer

index_offset As Integer

number As short

one As Integer

```

startingoffset As Integer

unknown() As Integer

unknown1\_count As Integer

unknown1\_offset As Integer

unknown2() As Integer

vertex\_count As Integer

vertex\_offset As Integer

zero As Integer

zeros() As Integer

zeros2() As Integer

End Class

## **Module Model\_module**

### **Model\_module.return\_mod2\_submodel:**

Function return\_mod2\_submodel(f as folderItem, minIndexOffset as integer, maxIndexOffset as integer, minVertOffset as integer, maxVertOffset as integer) As trimesh

if f <> nil then

dim br as BinaryStream = f.OpenAsBinaryFile  
br.LittleEndian = true  
br.Position = minVertOffset

Dim v\_count as integer = 0  
Dim v\_list() as Vector3D  
Dim normal\_list() as Vector3D  
Dim bi\_normal\_list() as Vector3D  
Dim tangent\_list() as Vector3D  
Dim uv\_u\_list() as single  
Dim uv\_v\_list() as single  
Dim unk2() as Vector3D

while br.Position <= maxVertOffset  
dim temp\_v as new Vector3D  
temp\_v.X = br.ReadSingle  
temp\_v.Y = br.ReadSingle  
temp\_v.Z = br.ReadSingle  
v\_list.Append(temp\_v)

dim temp\_v2 as new Vector3D  
temp\_v2.X = br.ReadSingle  
temp\_v2.Y = br.ReadSingle

```

temp_v2.Z = br.ReadSingle
normal_list.Append(temp_v2)

dim temp_v3 as new Vector3D
temp_v3.X = br.ReadSingle
temp_v3.Y = br.ReadSingle
temp_v3.Z = br.ReadSingle
bi_normal_list.Append(temp_v3)

dim temp_v4 as new Vector3D
temp_v4.X = br.ReadSingle
temp_v4.Y = br.ReadSingle
temp_v4.Z = br.ReadSingle
tangent_list.Append(temp_v4)

uv_u_list.Append br.ReadSingle
uv_v_list.Append br.ReadSingle

dim temp_v5 as new Vector3D
temp_v5.X = br.ReadSingle
temp_v5.Y = br.ReadSingle
temp_v5.Z = br.ReadSingle
unk2.Append(temp_v5)

v_count = v_count+1
wend

//now for the actual faces
dim f_count as integer = 0
dim f_listA() as integer
dim f_listB() as integer
dim f_listC() as integer
dim lines as integer = 0
br.Position = minIndexOffset //+ 2
while br.Position <= maxIndexOffset
    dim a_temp, b_temp, c_temp as int16
    //do
    a_temp = br.ReadInt16
    //loop until a_temp >= 0 AND a_temp < v_count
    //do
    'b_temp = br.ReadInt16
    b_temp = 0
    //loop until b_temp >= 0 AND b_temp < v_count
    //do
    'c_temp = br.ReadInt16
    c_temp = 0
    //loop until c_temp >= 0 AND c_temp < v_count

    'if a_temp = b_temp or b_temp = c_temp or a_temp = c_temp then
    'lines = lines + 1
    'end

    if a_temp = -1 or b_temp = -1 or c_temp = -1 or _

```

```

    a_temp >= v_count or b_temp >= v_count or c_temp >= v_count then
    //there's something weird about this indexing scheme
    //something is there that means start from x-th vert
    //br.position = maxIndexOffset + 4

    //perhaps there's something special about this FFFFF
else
    f_listA.Append a_temp
    'f_listB.Append b_temp
    'f_listC.Append c_temp
    f_count = f_count + 1
end
wend

//Dim return_tm as trimesh
return_tm = new trimesh

return_tm.renderbackfaces = true
return_tm.NullShader = true

//add the verts
dim a as integer = ubound(v_list) + 1
return_tm.VertexCount = a
for i as integer = 0 to ubound(v_list)
    return_tm.VertexPositions.SetVector(i, v_list(i))
next

return_tm.HasVertexNormals = true
for i as integer = 0 to ubound(normal_list)
    return_tm.VertexNormals.SetVector(i, normal_list(i))
next

return_tm.HasVertexUVs = true
for i as integer = 0 to ubound(uv_u_list)
    return_tm.VertexUVs.SetUV(i, uv_u_list(i), uv_v_list(i))
next
'break

//now add the faces as indicies
'dim c as integer = ubound(f_listA) + 1
dim c as integer = ubound(f_listA) - 1
return_tm.TriangleCount = c+2
for i as integer = 0 to c-1
    dim t as new vector3d
    t.x = v_list(f_listA(i)).x
    t.y = v_list(f_listA(i+1)).y
    t.z = v_list(f_listA(i+2)).z
    'return_tm.Triangles.SetABC(i, f_listA(i), f_listB(i), f_listC(i))
    return_tm.Triangles.SetABC(i, f_listA(i), f_listA(i+1), f_listA(i+2))

    //debugging
    //return_tm.Triangles.SetABC(i, 1, 2, 3)
next

```

```

'//add the last two triangles
'return_tm.Triangles.SetABC(c, f_listA(c), f_listA(c+1), f_listA(0))
'return_tm.Triangles.SetABC(c+1, f_listA(c+1), f_listA(0), f_listA(1))

//break
br.close

return return_tm
end

```

```

Exception
break
End Function
return_tm As trimesh

```

```
End Module
```

## **Class LODS\_submodels**

```
submodels(-1) As trimesh
```

```
End Class
```

## **Class Model\_Bones**

### **Model\_Bones.read:**

```

Sub read(byref br as binaryStream)
    dim offset as integer = br.position
    br.littleEndian = true

    Dim temp_str As string = ""
    Dim inputint as int8
    inputint = br.readint8
    br.position = br.position - 1
    while (not(inputint = 0))
        temp_str = temp_str + br.read(1)
        inputint = br.readint8
        br.position = br.position - 1
    wend
    name = temp_str

    br.Position = offset + 32
    NextSiblingNode = br.ReadShort
    NextChildNode = br.ReadShort
    ParentNode = br.ReadShort
    unk1 = br.ReadShort

    redim Translation(-1)
    for i as integer = 0 to 2
        Translation.append br.ReadSingle
    next

```



```

redim Rotation(-1)
for i as integer = 0 to 3
    Rotation.append br.ReadSingle
next

DistanceFromParent = br.ReadSingle

dim temp_position as integer = br.Position
br.Position = br.Position + 32
modelscale_position = br.Position
modelscale1 = br.ReadSingle
modelscale2 = br.ReadSingle

br.Position = temp_position
for i as integer = 1 to 3
    for j as integer = 1 to 7
        unk2(i,j) = br.ReadSingle
    next
next
End Sub
DistanceFromParent As single

modelscale1 As single

modelscale2 As single

modelscale_position As Integer

name As string

NextChildNode As short

NextSiblingNode As short

ParentNode As short

Rotation() As single

Translation() As single

unk1 As short

unk2(3,7) As single

End Class

```

## **Class Model Attachment Header**

### **Model\_Attachment\_Header.read:**

```

Sub read(byref br as binaryStream, magic as integer)
    dim offset as integer = br.position

```

```

br.littleEndian = true

Dim temp_str As string = ""
Dim inputint as int8
inputint = br.readint8
br.position = br.position - 1
while (not(inputint = 0))
    temp_str = temp_str + br.read(1)
    inputint = br.readint8
    br.position = br.position - 1
wend
name = temp_str

dim temp as new reflexive
br.Position = offset + &h34
chunk = temp.read(br)

redim points(-1)
for i as integer = 0 to chunk.count-1
    dim temp_point as new Model_Attachment_Point
    br.Position=chunk.offset - magic + (i*32)
    temp_point.read(br)
    points.Append temp_point
next
End Sub
chunk As reflexive

name As string

points() As Model_attachment_point

End Class

```

## **Class Model\_Attachment\_Point**

### **Model\_Attachment\_Point.read:**

```

Sub read(byref br as binaryStream)
    br.LittleEndian = true

    unknown = br.readint32

    dim temp as single

    redim translation(-1)
    for i as integer = 0 to 2
        temp = br.ReadSingle
        translation.Append temp
    next

    redim rotation(-1)
    for i as integer = 0 to 3
        temp = br.ReadSingle
    next

```

```

        rotation.Append temp
    next
End Sub
rotation() As single

translation() As single

unknown As Integer

End Class

```

## **Class Model**

### **Model.read:**

```

Sub read(f as folderitem, magic as integer, offset as integer, index_offset as integer, vertex_offset as integer)
    dim br as BinaryStream = f.openasbinaryfile
    br.littleendian = true
    br.position = offset
    dim temp_header as new Model_Header
    temp_header.read(br)
    header = temp_header

    iOffset = index_offset
    vOffset = vertex_offset

    redim model_chunk(-1)
    for i as integer = 0 to header.models.count-1
        dim temp as new Model_Reflexive_data
        br.Position = header.models.offset - magic + (76*i)
        temp.read(br, magic)
        model_chunk.append(temp)
    next

    'break
    redim bones(-1)
    for i as integer = 0 to header.bones.count-1
        dim temp as new Model_bones
        br.position = header.bones.offset - magic + (156*i)
        temp.read(br)
        bones.Append temp
    next

    redim attachment_points(-1)
    for i as integer = 0 to header.attachmentPoints.count-1
        dim temp as new Model_Attachment_Header
        br.Position = header.attachmentPoints.offset - magic + (64*i)
        temp.read(br, magic)
        attachment_points.Append temp
    next

    redim submodel(-1) //as SubModel_Header

```

```

dim maxTriangleStrips as integer = 0
dim TriangleStrip(-1, -1) as Model_TriangleStrip_Header

dim soffset as integer = header.submodels.offset - magic
soffset = soffset + 36
Dim metaStartOffset as integer

For x as integer = 0 to header.submodels.count - 1
    br.position = soffset
    dim temp as new SubModel_Header
    temp.read(br)
    submodel.Append temp
    soffset = br.Position

    if SubModel(x).count > 0 then
        br.Position = SubModel(x).offset - magic

        if SubModel(x).count > maxTriangleStrips then maxTriangleStrips = SubModel(x).count
        redim TriangleStrip(header.submodels.count - 1, maxTriangleStrips - 1)
        For y as integer = 0 to SubModel(x).count - 1
            dim tri as new Model_TriangleStrip_Header
            tri.read(br)
            TriangleStrip(x,y) = tri
        next
    end
next

Dim minVertexOffset as integer = &hFFFFFFF
Dim maxVertexOffset as integer = 0
Dim minIndexOffset as integer = &hFFFFFFF
Dim maxIndexOffset as integer = 0

maxIndexOffset = TriangleStrip(0, SubModel(0).count-1).index_offset
maxIndexOffset = maxIndexOffset + (2 * (TriangleStrip(0, SubModel(0).count - 1).index_count))
minIndexOffset = TriangleStrip(header.SubModels.count - 1, _
SubModel(header.SubModels.count - 1).count - 1).index_offset

maxVertexOffset = TriangleStrip(0, SubModel(0).count - 1).vertex_offset

maxVertexOffset = maxVertexOffset + (68 * (TriangleStrip(0, SubModel(0).count - 1).vertex_count - 1))

minVertexOffset = TriangleStrip(header.SubModels.count - 1, _
SubModel(header.SubModels.count - 1).count - 1).vertex_offset

minVertexOffset = minVertexOffset + vertex_offset
maxVertexOffset = maxVertexOffset + vertex_offset
minIndexOffset = minIndexOffset + index_offset + vertex_offset
maxIndexOffset = maxIndexOffset + index_offset + vertex_offset

MIN_INDEX_OFFSET = minIndexOffset
MAX_INDEX_OFFSET = maxIndexOffset
MIN_VERTEX_OFFSET = minVertexOffset
MAX_VERTEX_OFFSET = maxVertexOffset

```

```

redim meshes(-1)
for i as integer = 0 to header.SubModels.count - 1
    dim temp_LOD as new LODS_submodels
    for j as integer = 0 to SubModel(i).count-1
        maxIndexOffset = TriangleStrip(i, j).index_offset
        maxIndexOffset = maxIndexOffset + (2 * (TriangleStrip(i, j).index_count))
        minIndexOffset = TriangleStrip(i, j).index_offset

        maxVertexOffset = TriangleStrip(i, j).vertex_offset

        maxVertexOffset = maxVertexOffset + (68 * (TriangleStrip(i, j).vertex_count - 1))

        minVertexOffset = TriangleStrip(i, j).vertex_offset

        minVertexOffset = minVertexOffset + vertex_offset
        maxVertexOffset = maxVertexOffset + vertex_offset
        minIndexOffset = minIndexOffset + index_offset + vertex_offset
        maxIndexOffset = maxIndexOffset + index_offset + vertex_offset

        dim temp_tri as trimesh
        temp_tri = return_mod2_submodel(f, minIndexOffset, maxIndexOffset, minVertexOffset, maxVertexOffset)
        temp_LOD.submodels.Append temp_tri

    next
    meshes.Append temp_LOD
next

//only for me to do some stuff
'if true then
'dim f as FolderItem = GetOpenFolderItem(FileTypes1.HaloMapFile)
'if f = nil then return
'dim bw as BinaryStream = f.OpenAsBinaryFile(true)
'bw.LittleEndian = true
'for i as integer = 0 to UBound(bones)
'bw.Position = bones(i).modelscale_position
'bw.WriteSingle(0.5)
'next
'bw.close
'end
Exception
    break
End Sub
attachment_points() As Model_Attachment_Header

bones() As model_bones

header As model_Header

iOffset As Integer

MAX_INDEX_OFFSET As Integer

```

```

MAX_VERTEX_OFFSET As Integer

meshes() As LoDS_submodels

MIN_INDEX_OFFSET As Integer

MIN_VERTEX_OFFSET As Integer

model_chunk() As model_reflexive_data

submodel() As subModel_Header

vOffset As Integer

End Class

```

## **Class Reflexive**

### **Reflexive.read:**

```

Function read(ByRef br as binaryStream) As Reflexive
    br.LittleEndian = true
    dim temp as new reflexive
    temp.count = br.readint32
    temp.offset = br.readint32
    temp.unknown = br.readint32

    return temp
End Function
count As Integer

offset As Integer

unknown As Integer

End Class

```

## **Class type\_list**

```

class2 As string

idents() As Integer

tags() As string

End Class

```

## **Module FileManip\_old**

FileManip\_old.hex\_out:

Function hex\_out(value as integer) As string

dim output as string

output = hex(value)

select case output.len

case 0

//should never execute

output = "00000000" + output

case 1

output = "0000000" + output

case 2

output = "000000" + output

case 3

output = "00000" + output

case 4

output = "0000" + output

case 5

output = "000" + output

case 6

output = "00" + output

case 7

output = "0" + output

end

return output

End Function

### FileManip\_old.map\_output:

Function map\_output(map as mapfile) As string

'dim temp as new map\_dump\_thread

'temp.map = map

'temp.run

'dim progress as new progress\_dialog

'progress.show

'progress.init("Parsing Map Data", "Reading header...", map.metacount + 2)

'progress.update\_data(1)

'dim output as string = ""

'

'//header

'output = output + "=====\_"

'+ "===== " + EndOfLine

'output = output + "Header Data:" + EndOfLine + EndOfLine

'output = output + "Head string:" + " " + map.Headstring + EndOfLine

'output = output + "Game-type:" + " " + hex\_out(map.Gametype) + " " \_

'+ "05 xbox, 06 demo, 07 pc, 261 CE" + EndOfLine

'output = output + "Decompressed map size:" + " " + hex\_out(map.decompressedmapsize) \_

'+ " " + str(map.decompressedmapsize) + " bytes" + EndOfLine

'output = output + "Index offset:" + " " + hex\_out(map.indexoffset) + EndOfLine

'output = output + "Meta data size:" + " " + hex\_out(map.metastart) + EndOfLine

'output = output + "Map name:" + " " + map.mapname + EndOfLine

'output = output + "Map build date:" + " " + map.mapbuilddate + EndOfLine

```

'output = output + "Map type:" + " " + hex_out(map.maptype) + " " + "0 sp, 1 mp, 2 ui" + EndOfLine
'output = output + "Xor Checksum?:" + " " + hex_out(map.xorchecksum) + " " + "not verified" + EndOfLine
'output = output + "Foot string:" + " " + map.footstring + EndOfLine
'output = output + EndOfLine
,

'progress.update_data_text("Reading index header...", 2)
'//index header
'output = output + "=====_"
'+ "=====_" + EndOfLine
'output = output + "Index Header Data:" + EndOfLine + EndOfLine
'output = output + "Index magic:" + " " + hex_out(map.indexmagic) + EndOfLine
'output = output + "Tag count:" + " " + hex_out(map.metacount) + " " + str(map.metacount) + EndOfLine
'output = output + "Verticie count:" + " " + hex_out(map.verticecount) + EndOfLine
'output = output + "Verticie offset:" + " " + hex_out(map.modeldrawoffset) + EndOfLine
'output = output + "Indicie count:" + " " + hex_out(map.indiciecount) + EndOfLine
'output = output + "Indicie offset:" + " " + hex_out(map.modelindiceoffsets) + EndOfLine
'output = output + "Model Size:" + " " + hex_out(map.modelrawsize) + EndOfLine
'output = output + "Tag string:" + " " + map.tagstring + EndOfLine
'output = output + "Tag offset:" + " " + hex_out(map.tagoffset) + EndOfLine
'output = output + "Magic modifier:" + " " + hex_out(map.tempint) + EndOfLine
'output = output + "Map Magic:" + " " + hex_out(map.primarymagic) + EndOfLine
'output = output + EndOfLine
,
,

'//tag stuff
'progress.update_data_text("Reading tags...", 3)
'output = output + "=====_"
'+ "=====_" + EndOfLine
'output = output + "Tag Data:" + EndOfLine
,

'dim i as integer = 0
'while i < map.metacount
'progress.update_data(i + 3)
'output = output + EndOfLine
'output = output + "-----" + endlfline
'select case str(i+1).len
'case 0
'//should never execute
'output = output + "0000" + str(i+1)
'case 1
'output = output + "000" + str(i+1)
'case 2
'output = output + "00" + str(i+1)
'case 3
'output = output + "0" + str(i+1)
'case else
'output = output + str(i+1)
'end
'output = output + " " + map.tags_list(i).class2 + " " + map.tags_list(i).class2_2 + " " _
'+ map.tags_list(i).class2_3 + " " + map.tags_list(i).fullname + EndOfLine
'output = output + " " + "Identity:" + " " + hex_out(map.tags_list(i).ident) + " " _
'+ str(map.tags_list(i).ident) + EndOfLine
'output = output + " " + "String offset:" + " " + hex_out(map.tags_list(i).string_offset) + EndOfLine
'output = output + " " + "Meta offset:" + " " + hex_out(map.tags_list(i).offset) + EndOfLine

```



```

'output = output + "    " + "Estimated Size:" + "    " + hex_out(map.tags_list(i).estimated_meta_size) + EndOfLine
'i = i + 1
'wend
'

'output = output + "=====_"
'+ "=====_" + EndOfLine
'

'progress.close
'

'return output
End Function

```

### **FileManip\_old.return\_box:**

Function return\_box(x as single, y as single, z as single, scale as single = 0.01, grad as boolean = true) As trimesh  
 //so we do

```

//x + 0.001, y + 0.001, z + 0.001
//x + 0.001, y + 0.001, z - 0.001
//x + 0.001, y - 0.001, z + 0.001
//x + 0.001, y - 0.001, z - 0.001
//x - 0.001, y + 0.001, z + 0.001
//x - 0.001, y + 0.001, z - 0.001
//x - 0.001, y - 0.001, z + 0.001
//x - 0.001, y - 0.001, z - 0.001

```

dim return\_tm as new trimesh

//so lets add all those points

```

return_tm.VertexCount = 8
'return_tm.VertexPositions.SetXYZ(0, x + 0.001, y + 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(1, x + 0.001, y + 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(2, x + 0.001, y - 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(3, x + 0.001, y - 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(4, x - 0.001, y + 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(5, x - 0.001, y + 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(6, x - 0.001, y - 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(7, x - 0.001, y - 0.001, z - 0.001)
return_tm.VertexPositions.SetXYZ(0, x + scale, y + scale, z + scale)
return_tm.VertexPositions.SetXYZ(1, x + scale, y + scale, z - scale)
return_tm.VertexPositions.SetXYZ(2, x + scale, y - scale, z + scale)
return_tm.VertexPositions.SetXYZ(3, x + scale, y - scale, z - scale)
return_tm.VertexPositions.SetXYZ(4, x - scale, y + scale, z + scale)
return_tm.VertexPositions.SetXYZ(5, x - scale, y + scale, z - scale)
return_tm.VertexPositions.SetXYZ(6, x - scale, y - scale, z + scale)
return_tm.VertexPositions.SetXYZ(7, x - scale, y - scale, z - scale)

```

if grad then

```

return_tm.HasVertexColors = true
return_tm.VertexColors.Item(0) = &cFFFAFA
return_tm.VertexColors.Item(1) = &cFFFAFA
return_tm.VertexColors.Item(2) = &cFFFAFA
return_tm.VertexColors.Item(3) = &cFFFAFA
return_tm.VertexColors.Item(4) = &cFFFAFA
return_tm.VertexColors.Item(5) = &cFFFAFA

```

```

    return_tm.VertexColors.Item(6) = &cFFFAFA
    return_tm.VertexColors.Item(7) = &cFFFAFA
end

```

```

return_tm.HasVertexNormals = true
return_tm.VertexNormals.SetXYZ(0, 1, 1, 1)
return_tm.VertexNormals.SetXYZ(1, 1, 1, -1)
return_tm.VertexNormals.SetXYZ(2, 1, -1, 1)
return_tm.VertexNormals.SetXYZ(3, 1, -1, -1)
return_tm.VertexNormals.SetXYZ(4, -1, 1, 1)
return_tm.VertexNormals.SetXYZ(5, -1, 1, -1)
return_tm.VertexNormals.SetXYZ(6, -1, -1, 1)
return_tm.VertexNormals.SetXYZ(7, -1, -1, -1)

```

```

//now figure out all of the triangles necessary for rendering the box
//face in YZ-plane at x=1
//0, 1, 2
//1, 2, 3
//face in XZ-plane at y = 1
//0, 1, 5
//4, 0, 5
//face in YZ-plane at x = -1
//4, 5, 7
//7, 6, 4
//face in XZ-plane at y = -1
//2, 3, 7
//7, 6, 2
//face in XY-plane at z = 1
//2, 0, 4
//4, 6, 2
//face in XY-plane at z = -1
//3, 1, 5
//5, 7, 3

```

```

return_tm.TriangleCount = 12
return_tm.Triangles.SetABC(0, 0, 1, 2)
return_tm.Triangles.SetABC(1, 1, 2, 3)
return_tm.Triangles.SetABC(2, 0, 1, 5)
return_tm.Triangles.SetABC(3, 4, 0, 5)
return_tm.Triangles.SetABC(4, 4, 5, 7)
return_tm.Triangles.SetABC(5, 7, 6, 4)
return_tm.Triangles.SetABC(6, 2, 3, 7)
return_tm.Triangles.SetABC(7, 7, 6, 2)
return_tm.Triangles.SetABC(8, 2, 0, 4)
return_tm.Triangles.SetABC(9, 4, 6, 2)
return_tm.Triangles.SetABC(10, 3, 1, 5)
return_tm.Triangles.SetABC(11, 5, 7, 3)
return_tm.RenderBackFaces = true

```

```

    return return_tm
End Function
End Module

```

## **Class BOUNDING BOX**

### **BOUNDING\_BOX.read:**

Function read(ByRef br as binaryStream) As BOUNDING\_BOX

br.LittleEndian = true

for i as integer = 1 to 3

me.min(i) = br.ReadSingle

next

for i as integer = 1 to 3

me.max(i) = br.ReadSingle

next

End Function

max(3) As single

min(3) As single

End Class

### **Class tag**

class2 As string

class2\_2 As string

class2\_3 As string

estimated\_meta\_size As Integer

fullname As string

ident As Integer

### **tag.offset:**

offset As Integer

//modified by magic

### **tag.string\_offset:**

string\_offset As Integer

//modified by magic

zero1 As Integer

zero2 As Integer

### **tag Note: tag format**

tag format

class2

class2\_2

class2\_3

ident

```
stringoffset //this is saved modified by magic
metaoffset //this is saved modified by magic
```

End Class

## **Class mapfile**

### **mapfile.demo\_header\_load:**

```
Sub demo_header_load()
  dim br_local as binarystream
  br_local = f.openasbinaryfile
  br_local.littleEndian = true

  br_local.position = &h2c0
  Headstring = br_local.read(4)

  br_local.position = &h588
  Gametype = br_local.readint32

  br_local.position = &h5e8
  decompressedmapsize = br_local.readint32

  br_local.position = &h5ec
  indexoffset = br_local.readint32

  br_local.position = &h2c4
  metastart = br_local.readint32

  br_local.position = &h58c

  dim i as integer = 0
  while i < 32
    dim inputint as int8
    inputint = br_local.readint8

    if (not(inputint = 0))then
      br_local.position = br_local.position - 1
      mapname = mapname + br_local.read(1)
    end
    i = i + 1
  wend

  mapname_proper = get_name(mapname)

  br_local.position = &h2c8
  //Mapbuilddate = br_local.read(32)
  Mapbuilddate = br_local.read(13)

  br_local.position = &h02
  maptype = br_local.readshort
```

```
br_local.position = 100
xorchecksum = br_local.readint32
```

```
br_local.position = &h5f0
Footstring = br_local.read(4)
```

```
br_local.close
End Sub
```

### **mapfile.get\_name:**

```
Function get_name(mapname as string) As string
    dim result as string
```

```
    select case mapname
    Case "a10"
```

```
        result = "Pillar of Autum"
```

```
    Case "a30"
```

```
        result = "Halo"
```

```
    Case "a50"
```

```
        result = "Truth and Reconciliation"
```

```
    Case "b30"
```

```
        result = "Silent Cartographer"
```

```
    Case "b40"
```

```
        result = "Assault on the Control Room"
```

```
    Case "c10"
```

```
        result = "Guilty Spark"
```

```
    Case "c20"
```

```
        result = "Library"
```

```
    Case "c40"
```

```
        result = "Two Betrayals"
```

```
    Case "d20"
```

```
        result = "Keyes"
```

```
    Case "d40"
```

result = "The Maw"

Case "beavercreek"

result = "Battle Creek"

Case "bloodgulch"

result = "Blood Gulch"

Case "boardingaction"

result = "Boarding Action"

Case "carousel"

result = "Derelict"

Case "chillout"

result = "Chill Out"

Case "damnation"

result = "Damnation"

Case "hangemhigh"

result = "Hang 'em High"

Case "longest"

result = "Longest"

Case "prisoner"

result = "Prisoner"

Case "putput"

result = "Chiron TL34"

Case "ratrace"

result = "Rat Race"

Case "sidewinder"

result = "Sidewinder"

Case "wizard"

result = "Wizard"

```
Case "deathisland"
```

```
    result = "Death Island"
```

```
Case "dangercanyon"
```

```
    result = "Danger Canyon"
```

```
Case "gephyrophobia"
```

```
    result = "Gephyrophobia"
```

```
Case "timberland"
```

```
    result = "Timberland"
```

```
Case "icefields"
```

```
    result = "Ice Fields"
```

```
Case "infinity"
```

```
    result = "Infinity"
```

```
Case "ui"
```

```
    result = "User Interface"
```

```
case else
```

```
    result = mapname
```

```
    //demo testing
```

```
    if(mapname.instr("b30") > 0)then
```

```
        result = "Silent Cartographer"
```

```
    end
```

```
    if(mapname.instr("bloodgulch") > 0)then
```

```
        result = "Blood Gulch"
```

```
    end
```

```
    if(mapname.instr("ui")>0)then
```

```
        result = "User Interface"
```

```
    end
```

```
end
```

```
    return result
```

```
End Function
```

### **mapfile.load:**

Function load() As boolean

```
    br = f.openasbinaryfile
```

```
    br.LittleEndian = true
```

```

dim output as boolean = true
dim temp_name_dictionary as new dictionary
dim temp_ident_dictionary as new dictionary
redim tags_list(-1)

'dim Headstring as string
'dim Gametype as int32 //05 xbox, 06 demo, 07 pc, 609 CE
'dim decompressedmapsize as int32
'dim indexoffset as int32
'dim metastart as int32
'dim mapname as string
'dim mapbuilddate as string
'dim maptype as int32 // 0 = sp 1 = mp 2 = ui
'dim xorchecksum as int32
'dim footstring as string
'dim tempint as int32
'

//alternate header variables in order
'id
'version
'decomp_len
'zeros
'offset_to_index_decomp
'metadatasize
'zeros2[2]
'name[32]
'builddate[32]
'maptype
'id2

'dim indexMagic as int32
'dim verticecount as int32
'dim modelrawoffset as int32
'dim indiciecount as int32
'dim modelindiceoffsets as int32
'dim modelrawsize as int32
'dim tagsoffset as int32
'dim tagstring as string

//check which head if either to read
br.position = 0
dim head_read as string = br.read(4)
dim valid as boolean = false
if head_read = "daeh" then
    normal_header_load
    valid = true
else
    //possible demo map
    br.position = &h2C0
    head_read = br.read(4)
    if head_read = "dehE" then
        //valid demo header
        demo_header_load

```



```

    valid = true
end
end

if valid then

    br.position = indexoffset
    indexmagic = br.readInt32

    br.position = indexoffset + 4
    base_tag_id = br.readInt32

    //br.position = indexoffset + 8
    //vertexsize = br.readInt32

    br.position = indexoffset + 12
    metacount = br.readInt32
    //tagcount

    br.position = indexoffset + 16
    verticecount = br.readInt32
    //vertex_object_count

    br.position = indexoffset + 20
    modelrawoffset = br.readInt32
    //vertex_offset

    br.position = indexoffset + 24
    indiciecount = br.readInt32
    //indices_object_count

    br.position = indexoffset + 28
    modelindiceoffsets = br.readInt32
    //vertex_size

    br.position = indexoffset + 32
    modelrawsize = br.readInt32
    //model_size

    br.position = indexoffset + 36
    tagstring = br.read(4)
    //tagstart

    tagsoffset = indexOffset + 40

    tempint = indexOffset + 40

    //calculate the magic value for the map
    primaryMagic = indexMagic - tempint

    //Index Items

    dim i as integer

```

i = 0

While i < metacount

dim temp\_tag as new tag

//get the class2 primary

br.position = tagsoffset + (i \* 32) //+ 4

dim tempchar As string= br.Read(4, encodings.ISOLatin1)

tempchar = reverse(tempchar)

temp\_tag.class2 = tempchar

//get the class2 2nd part

br.position = tagsoffset + (i \* 32) + 4 //+ 4

tempchar = br.read(4, encodings.ISOLatin1)

tempchar = reverse(tempchar)

temp\_tag.class2\_2 = tempchar

//get the class2 3rd part

br.position = tagsoffset + (i \* 32) + 8 //+ 4

tempchar = br.read(4, encodings.ISOLatin1)

tempchar = reverse(tempchar)

temp\_tag.class2\_3 = tempchar

//get the identity

br.position = tagsoffset + (i \* 32) + 12 //+ 4

temp\_tag.ident = br.readint32

if temp\_tag.ident = base\_tag\_id then

base\_tag = i

end

//get the name and string offset

br.position = tagsoffset + (i \* 32) + 16 //+ 4

//quick little name finder

temp\_tag.string\_offset = br.readint32 - primarymagic

br.position = temp\_tag.string\_offset //+ 4

Dim temp As string = ""

Dim inputint as int8

inputint = br.readint8

br.position = br.position - 1

while (not(inputint = 0))

temp = temp + br.read(1)

inputint = br.readint8

br.position = br.position - 1

wend

temp\_tag.fullname = temp.convertEncoding(encodings.ascii)

//get the offset

br.position = tagsoffset + (i \* 32) + 20 //+ 4

temp\_tag.offset = br.readint32 - primaryMagic

//all of this is null data, was hoping to find the raw offset info

//get something

br.position = tagsoffset + (i \* 32) + 24 //+ 4

```

temp_tag.zero1 = br.readint32

//get something else
br.position = tagsoffset + (i * 32) + 28 //+ 4
temp_tag.zero2 = br.readint32

'if (something1 <> 0 or something2 <> 0)then
'msgbox("something1: " + str(something1) + EndOfLine + "something2: " + str(something2))
'end

//add the tag to the list
tags_list.append(temp_tag)

if(i > 0)then
    tags_list(i-1).estimated_meta_size = tags_list(i).offset - tags_list(i-1).offset
end

if i > 0 then
    if tags_list(i-1).class2 = "sbsp" then
        dim bsp_size as integer = 0
        dim a as Scenario_Header = scenario_header_read(me, base_tag)
        dim br2 as BinaryStream = f.OpenAsBinaryFile
        br2.LittleEndian = true
        br2.Position = a.StructBsp.offset - primarymagic
        dim BspSize() as UInt32
        dim unknown() as uint32
        dim tagid() as int32
        for j as integer = 1 to a.StructBsp.count
            unknown.append br2.readuint32
            BspSize.append br2.ReadUInt32
            unknown.Append br2.ReadUInt32
            unknown.Append br2.ReadUInt32
            unknown.append br2.readUInt32
            unknown.append br2.ReadUInt32
            unknown.Append br2.ReadUInt32
            tagid.Append br2.ReadInt32
        next
        for j as integer = 0 to ubound(tagid)
            if tags_list(i-1).ident = tagid(j) then
                bsp_size = bspsize(j)
                j = ubound(tagid)+1
            end
        next
        br2.close
        tags_list(i-1).estimated_meta_size = BSP_size
    end
end

//add the data to the dictionaries
temp = temp_tag.class2 + ":" + temp_tag.fullname
temp_name_dictionary.value(temp) = i
temp_ident_dictionary.value(temp_tag.ident) = i
dim new_type as boolean = true
for x as integer = 0 to ubound(displists)

```

```

        if(temp_tag.class2 = disp_list(x).class2)then
            disp_list(x).tags.append temp_tag.fullname
            disp_list(x).idents.append temp_tag.ident
            x = ubound(disp_list)
            new_type = false
        end
    next
    if new_type then
        dim temp_type as new type_list
        temp_type.class2 = temp_tag.class2
        temp_type.idents.append temp_tag.ident
        temp_type.tags.append temp_tag.fullname
        disp_list.append temp_type
    end

    i = i+1

wend

i = 0

tags_by_class_name = temp_name_dictionary
tags_by_ident = temp_ident_dictionary

else

    errorbox("Invalid file data")
    output = false

end

br.close

return output
End Function

```

### **mapfile.normal\_header\_load:**

```

Sub normal_header_load()
    dim br_local as binarystream
    br_local = f.openasbinaryfile
    br_local.littleEndian = true

    br_local.position = 0
    Headstring = br_local.read(4)

    br_local.position = 4
    Gametype = br_local.readint32
    if gametype = 609 then
        ce = true
    end

    br_local.position = 8
    decompressedmapsize = br_local.readint32

```

```

br_local.position = 16
indexoffset = br_local.readint32

br_local.position = 20
metastart = br_local.readint32

br_local.position = 32

dim i as integer = 0
while i < 32
    dim inputint as int8
    inputint = br_local.readint8

    if (not(inputint = 0))then
        br_local.position = br_local.position - 1
        mapname = mapname + br_local.read(1)
    end
    i = i + 1
wend

mapname_proper = get_name(mapname)

br_local.position = 64
//Mapbuilddate = br_local.read(32)
Mapbuilddate = br_local.read(13)

br_local.position = 96
maptype = br_local.readshort

br_local.position = 100
xorchecksum = br_local.readint32

br_local.position = 2044
Footstring = br_local.read(4)

br_local.close
End Sub

```

### **mapfile.reverse:**

Function reverse(input as string) As string

//this function reverses the characters it receives (which it will call input) and reverses the order

//find the length of the string passed

dim i as integer = Len(input)

dim output as string

//start from the last character in the string and write it to the string output

//then decrement i, to read the next earlier character

while i >=0

//the mid function starts from the i-th character, and gets in this case 1 character

```

        output = output + input.mid(i,1)
        i=( i - 1)
    wend

    //this returns the reversed string to anything calling it
    //this statement is extremely important for all you non-programmers out there
    return output
End Function
base_tag As Integer

base_tag_id As Integer

br As binarystream

CE As boolean

decompressedmapsize As int32

disp_list() As type_list

f As folderitem

footstring As string

Gametype As int32

Headstring As string

indexmagic As int32

indexoffset As int32

indiciecount As int32

mapbulddate As string

mapname As string

mapname_proper As string

maptype As int32

metacount As Integer

mapfile.metastart:
metastart As int32
//also refered to as meta byte count
modelindiceoffsets As int32

modelrawoffset As int32

modelrawsize As int32

```

primarymagic As Integer

tagoffset As int32

tagstring As string

tags\_by\_class\_name As dictionary

tags\_by\_ident As dictionary

tags\_list() As tag

tempint As Integer

verticecount As int32

xorchecksum As int32

End Class

## **Class Dependency**

### **Dependency.read:**

Function read(ByRef br as binaryStream) As Dependency

    dim temp as new Dependency

    br.LittleEndian = true

    temp.tag = reverse(br.read(4))

    temp.NamePtr = br.readint32

    temp.unknown = br.readint32

    temp.TagId = br.readint32

    return temp

End Function

NamePtr As int32

tag As string

TagId As int32

unknown As int32

End Class

## **Class Model\_window**

Inherits Window

### **Model\_window.CancelClose:**

Function CancelClose(appQuitting as Boolean) As Boolean

    if not appQuitting then

        me.Visible = false

```

        return false
    end
End Function

```

### **Model\_window.Close:**

```

Sub Close()
    'for i as integer = 0 to Rb3DSpace1.Objects.Count
    'Rb3DSpace1.Objects.Remove(0)
    'next
    '

    'Rb3DSpace1.Objects = nil
    'me.Rb3DSpace1.close

    me.Visible = false
End Sub

```

### **Model\_window.Open:**

```

Sub Open()
    #if DebugBuild
        skin_push.Visible = true
        skin_push.Enabled = true
    #else
        skin_push.Visible = false
        skin_push.Enabled = false
    #endif
End Sub

```

### **Model\_window.menuitemclose:**

```

Function menuitemclose() As Boolean
    me.Visible = false
End Function

```

### **Model\_window.controls\_update:**

```

Sub controls_update()
    dim x as double = Rb3DSpace1.Camera.Position.X
    dim y as double = Rb3DSpace1.Camera.Position.Y
    dim z as double = Rb3DSpace1.Camera.Position.Z

    //the if loops are what allow the slider to center once it has reached an end point
    x_slider.Value=x*10
    if x_slider.Value = x_slider.Maximum then
        x_slider.Minimum = x_slider.Minimum + 300
        x_slider.Maximum = x_slider.Maximum + 300
    end
    if x_slider.Value = x_slider.Minimum then
        x_slider.Minimum = x_slider.Minimum - 300
        x_slider.Maximum = x_slider.Maximum - 300
    end

    y_slider.Value=y*10
    if y_slider.Value = y_slider.Maximum then
        y_slider.Minimum = y_slider.Minimum + 300
        y_slider.Maximum = y_slider.Maximum + 300
    end

```



```

end
if y_slider.Value = y_slider.Minimum then
    y_slider.Minimum = y_slider.Minimum - 300
    y_slider.Maximum = y_slider.Maximum - 300
end

z_slider.Value=z*10
if z_slider.Value = z_slider.Maximum then
    z_slider.Minimum = z_slider.Minimum + 300
    z_slider.Maximum = z_slider.Maximum + 300
end
if z_slider.Value = z_slider.Minimum then
    z_slider.Minimum = z_slider.Minimum - 300
    z_slider.Maximum = z_slider.Maximum - 300
end

x_edit.text = str(x)
y_edit.text = str(y)
z_edit.text = str(z)

pitch_edit.text = str(pitch_slider.value / 100.0)
yaw_edit.text = str(yaw_slider.value / 100.0)
roll_edit.text = str(roll_slider.value / 100.0)

Rb3DSpace1.Update
End Sub

```

### **Model\_window.init:**

```

Sub init(in_model as model, in_f as folderitem)
    model1 = in_model
    f = in_f

    redim meshes(-1)
    for i as integer = 0 to ubound(in_model.meshes)
        meshes.Append in_model.meshes(i)
    next

    Rb3DSpace1.Wireframe = true
    Rb3DSpace1.DebugCube = false
    roll_slider.Enabled = True
    yaw_slider.Enabled = True
    pitch_slider.Enabled = True
    x_slider.Value = 0
    y_slider.Value = 0
    z_slider.Value = 0
    control_tabs.value = 0
End Sub

```

### **Model\_window.return\_box:**

```

Protected Function return_box(x as single, y as single, z as single) As trimesh
    //so we do

    //x + 0.001, y + 0.001, z + 0.001
    //x + 0.001, y + 0.001, z - 0.001

```

```
//x + 0.001, y - 0.001, z + 0.001
//x + 0.001, y - 0.001, z - 0.001
//x - 0.001, y + 0.001, z + 0.001
//x - 0.001, y + 0.001, z - 0.001
//x - 0.001, y - 0.001, z + 0.001
//x - 0.001, y - 0.001, z - 0.001
```

```
dim return_tm as new trimesh
```

```
//so lets add all those points
return_tm.VertexCount = 8
'return_tm.VertexPositions.SetXYZ(0, x + 0.001, y + 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(1, x + 0.001, y + 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(2, x + 0.001, y - 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(3, x + 0.001, y - 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(4, x - 0.001, y + 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(5, x - 0.001, y + 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(6, x - 0.001, y - 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(7, x - 0.001, y - 0.001, z - 0.001)
return_tm.VertexPositions.SetXYZ(0, x + 0.01, y + 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(1, x + 0.01, y + 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(2, x + 0.01, y - 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(3, x + 0.01, y - 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(4, x - 0.01, y + 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(5, x - 0.01, y + 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(6, x - 0.01, y - 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(7, x - 0.01, y - 0.01, z - 0.01)
```

```
return_tm.HasVertexColors = true
return_tm.VertexColors.Item(0) = &cFFFAFA
return_tm.VertexColors.Item(1) = &cFFFAFA
return_tm.VertexColors.Item(2) = &cFFFAFA
return_tm.VertexColors.Item(3) = &cFFFAFA
return_tm.VertexColors.Item(4) = &cFFFAFA
return_tm.VertexColors.Item(5) = &cFFFAFA
return_tm.VertexColors.Item(6) = &cFFFAFA
return_tm.VertexColors.Item(7) = &cFFFAFA
```

```
return_tm.HasVertexNormals = true
return_tm.VertexNormals.SetXYZ(0, 1, 1, 1)
return_tm.VertexNormals.SetXYZ(1, 1, 1, -1)
return_tm.VertexNormals.SetXYZ(2, 1, -1, 1)
return_tm.VertexNormals.SetXYZ(3, 1, -1, -1)
return_tm.VertexNormals.SetXYZ(4, -1, 1, 1)
return_tm.VertexNormals.SetXYZ(5, -1, 1, -1)
return_tm.VertexNormals.SetXYZ(6, -1, -1, 1)
return_tm.VertexNormals.SetXYZ(7, -1, -1, -1)
```

```
//now figure out all of the triangles necessary for rendering the box
//face in YZ-plane at x=1
//0, 1, 2
//1, 2, 3
//face in XZ-plane at y = 1
//0, 1, 5
```

```

//4, 0, 5
//face in YZ-plane at x = -1
//4, 5, 7
//7, 6, 4
//face in XZ-plane at y = -1
//2, 3, 7
//7, 6, 2
//face in XY-plane at z = 1
//2, 0, 4
//4, 6, 2
//face in XY-plane at z = -1
//3, 1, 5
//5, 7, 3

return_tm.TriangleCount = 12
return_tm.Triangles.SetABC(0, 0, 1, 2)
return_tm.Triangles.SetABC(1, 1, 2, 3)
return_tm.Triangles.SetABC(2, 0, 1, 5)
return_tm.Triangles.SetABC(3, 4, 0, 5)
return_tm.Triangles.SetABC(4, 4, 5, 7)
return_tm.Triangles.SetABC(5, 7, 6, 4)
return_tm.Triangles.SetABC(6, 2, 3, 7)
return_tm.Triangles.SetABC(7, 7, 6, 2)
return_tm.Triangles.SetABC(8, 2, 0, 4)
return_tm.Triangles.SetABC(9, 4, 6, 2)
return_tm.Triangles.SetABC(10, 3, 1, 5)
return_tm.Triangles.SetABC(11, 5, 7, 3)
return_tm.RenderBackFaces = true

return return_tm
End Function

Model_window.setcolor:
Private Function setcolor(in_color as color) As picture
    //thanks to G3David for providing this code
    //because I don't like Rb3DSpaces, RB all that much and hate bitmaps
    //oh how I hate you bitmaps
    //much hate...
    //so much hate...
    //btw, I haven't started on it yet so this is all premature anxiety but still...
    //...
    //hate

    dim p as new Picture(1,1,32)
    p.Graphics.ForeColor=in_Color
    p.Graphics.DrawRect (0,0,1,1)
    return p
End Function

```

### **Model\_window.show\_attachment\_point:**

```

Protected Sub show_attachment_point(i_index as integer, j_index as integer)
    wireframe_check.Value = false

    yaw_slider.Value = 0

```

```

roll_slider.Value = 0
pitch_slider.Value = 0
controls_update
redim display_meshes(-1)
while Rb3DSpace1.Objects.Count > 0
    Rb3DSpace1.Objects.Remove(0)
wend

//so first display a single LoD (super high!)
for i as integer = 0 to ubound(Model1.model_chunk)
    for j as integer = 0 to ubound(Model1.meshes( Model1.model_chunk(i).links(0).super_high ).submodels)
        display_meshes.Append(Model1.meshes( Model1.model_chunk(i).links(0).super_high ).submodels(j))
    next
next

//now display the selected attachment point
display_meshes.Append( return_box( _
model1.attachment_points(i_index).points(j_index).translation(0) , _
model1.attachment_points(i_index).points(j_index).translation(1) , _
model1.attachment_points(i_index).points(j_index).translation(2) _
) )

attach_x_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(0))
attach_y_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(1))
attach_z_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(2))

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next

Rb3DSpace1.Update
End Sub

```

### **Model\_window.show\_attachment\_points:**

```

Protected Sub show_attachment_points()
    //first it must delete all visible meshes
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    //create a list of all the possible attachment points
    //listbox 2nd column: attachment_point index, listbox 3rd column: attachment_point point index

    attach_list.DeleteAllRows
    for i as integer = 0 to ubound(model1.attachment_points)
        for j as integer = 0 to ubound(model1.attachment_points(i).points)
            if ubound(model1.attachment_points(i).points) > 0 then
                attach_list.AddRow(model1.attachment_points(i).name + ": " + str(1+j))
            else

```

```

        attach_list.AddRow(model1.attachment_points(i).name)
    end
    attach_list.Cell(attach_list.LastIndex, 1) = str(i)
    attach_list.Cell(attach_list.LastIndex, 2) = str(j)
next
next

Rb3DSpace1.Update

//and initialize to the first element
//if there aren't any then ignore this
if attach_list.listcount > 0 then
    attach_list.Selected(0) = true
    show_attachment_point(0,0)
end
End Sub

```

### **Model\_window.show\_bone:**

Protected Function show\_bone(bone\_index as integer) As trimesh  
 //the other way to read all bones  
 //just add together all the parent bones coords

```

dim this_index as integer = bone_index
dim x as single = 0
dim y as single = 0
dim z as single = 0

while this_index > -1
    x = x + model1.bones(this_index).Translation(0)
    y = y + model1.bones(this_index).Translation(1)
    z = z + model1.bones(this_index).Translation(2)
    this_index = model1.bones(this_index).ParentNode
wend

dim tri as new Trimesh

tri = return_box( x, y, z )
tri.Material.Texture = setcolor(&cFFFFFF)
return tri
End Function

```

### **Model\_window.show\_bones:**

```

Protected Sub show_bones()
    wireframe_check.Value = false

    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

```

```

bones_list.DeleteAllRows
bones_list.ColumnAlignment(3) = 3
bones_list.ColumnAlignment(4) = 3
bones_list.ColumnAlignment(5) = 3
for i as integer = 0 to ubound(model1.bones)
    if i < 10 then
        bones_list.AddRow("0" + str(i))
    else
        bones_list.AddRow(str(i))
    end
    bones_list.Cell(bones_list.LastIndex, 1) = Model1.bones(i).name
    if model1.bones(i).ParentNode > -1 then
        bones_list.Cell(bones_list.LastIndex, 2) = Model1.bones( _
            model1.bones(i).ParentNode ).name
    else
        bones_list.Cell(bones_list.LastIndex, 2) = "None"
    end
    bones_list.Cell(bones_list.LastIndex, 3) = str(Model1.bones(i).Translation(0))
    bones_list.Cell(bones_list.LastIndex, 4) = str(Model1.bones(i).Translation(1))
    bones_list.Cell(bones_list.LastIndex, 5) = str(Model1.bones(i).Translation(2))
next

dim bones(-1) as Trimesh
for i as integer = 0 to UBound(model1.bones)
    bones.Append show_bone(i)
next

for i as integer = 0 to ubound(bones)
    display_meshes.Append(bones(i))
next
for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next
Rb3DSpace1.Update

bones_list.Selected(0) = true
show_selected_bone(0)
End Sub

```

### **Model\_window.show\_selected\_bone:**

```

Protected Sub show_selected_bone(actual_index as integer)
    //this method takes an index and alters the color of the correct bone

    //erase the current space
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend
    //white out all the current bones
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Material.Texture = setcolor(&cFFFFFF)
    next
    //for the alt method
    display_meshes(actual_index).Material.Texture = setcolor(&c00FF00)
    //render

```

```

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next
Rb3DSpace1.Update
End Sub

```

### **Model\_window.show\_sub\_model:**

```

Protected Sub show_sub_model(mesh_index as integer, submodel_index as integer)
    //I really don't know why I'm making this a method but I am
    //so screw you whoever is reading this and making fun of my coding
    //I went and had to research how everything works using only HMT and SparkEdit source codes
    //for the record, obj-c is an ugly language that deserves to be destroyed.
    //yes...destroyed. double past tense.
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update

    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    redim display_meshes(-1)
    display_meshes.append model1.meshes(mesh_index).submodels(submodel_index)

    for i as integer = 0 to ubound(display_meshes)
        Rb3DSpace1.Objects.Append display_meshes(i)
    next

    Rb3DSpace1.Update
End Sub

```

### **Model\_window.show\_sub\_models:**

```

Protected Sub show_sub_models()
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    Rb3DSpace1.Update

    sub_model_popup.DeleteAllRows
    sub_model_list.DeleteAllRows

    for i as integer = 0 to ubound(model1.meshes)
        //this is the first version. for a later version consider implementing
        if i < 10 then
            sub_model_popup.AddRow("Mesh #0" + str(i))
        else
            sub_model_popup.AddRow("Mesh #" + str(i))
        end if
    next
end Sub

```

```

    end
next

    sub_model_popup.ListIndex = 0
    //the selected list index will then trigger another bit of code relating to the listbox
End Sub
display_meshes() As trimesh

Protected f As folderitem

meshes() As LoDS_submodels

model1 As model

```

## **Model\_window Control Rb3DSpace1:**

```

Sub Open()
    if me.Objects = nil then
        errorbox("You need to install Quesa")
    end

    //set camera position (looking backwards)
    'me.camera.position.Z = 1
    'me.camera.position.X = 0
    //me.camera.pitch 3.1415926536
    //me.camera.roll 3.1415926536

    //Rb3DSpace1.Update
End Sub

```

## **Model\_window Control control\_tabs:**

```

Sub Change()

    select case me.value

    case 0
        //model data
        LOD_popup.DeleteAllRows

        LOD_popup.addrow("Super High")
        LOD_popup.addrow("High")
        LOD_popup.addrow("Medium")
        LOD_popup.addrow("Low")
        LOD_popup.addrow("Super Low")

        LOD_popup.ListIndex = 0
    case 1
        //SubModels
        show_sub_models
    case 2
        //bones
        show_bones
    end select

```



```

case 3
    //attachment points
    show_attachment_points

end select

me.refresh
End Sub
Model_window Control wireframe_check:

```

```

Sub Action()
    if me.Value = true then
        Rb3DSpace1.Wireframe = true
        Rb3DSpace1.Update
    else
        Rb3DSpace1.Wireframe = false
        Rb3DSpace1.Update
    end
End Sub

```

### **Model\_window Control x\_slider:**

```

Sub ValueChanged()
    Rb3DSpace1.Camera.Position.X = me.Value/10

    controls_update
End Sub

```

### **Model\_window Control x\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        x_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function

```

### **Model\_window Control y\_slider:**

```

Sub ValueChanged()
    Rb3DSpace1.Camera.Position.Y = me.Value/10

    controls_update
End Sub

```

### **Model\_window Control y\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        y_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function

```

```

    end
End Function

Model_window Control z_slider:

Sub ValueChanged()
    Rb3DSpace1.Camera.Position.Z = me.Value/10

```

```

    controls_update
End Sub

```

### **Model\_window Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        z_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function

```

### **Model\_window Control pitch\_slider:**

```

Sub ValueChanged()
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Pitch((val(pitch_edit.text) - (me.Value / 100.0)) * 3.1415926536)
    next

```

```

    controls_update
End Sub

```

### **Model\_window Control pitch\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim old_val as integer = pitch_slider.Value/100.0
        dim value as single = val(me.Text) * 100.0
        //psuedo modulo for dealing with single
        while value > 200.0
            value = value - 200.0
        wend
        while value < -200.0
            value = value + 200.0
        wend
        for i as integer = 0 to ubound(display_meshes)
            display_meshes(i).Pitch((old_val - (value / 100.0)) * 3.1415926536)
        next
        pitch_slider.Value = value

```

```

    controls_update
    return true
end
End Function

```

### **Model\_window Control roll\_slider:**

```

Sub ValueChanged()
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Roll((val(roll_edit.text) - (me.Value / 100.0)) * 3.1415926536)
    next
    controls_update
End Sub

```

### **Model\_window Control roll\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim old_val as integer = roll_slider.Value/100.0
        dim value as single = val(me.Text) * 100.0
        //psuedo modulo for dealing with single
        while value > 200.0
            value = value - 200.0
        wend
        while value < -200.0
            value = value + 200.0
        wend
        for i as integer = 0 to ubound(display_meshes)
            display_meshes(i).Roll((old_val - (value / 100.0)) * 3.1415926536)
        next
        roll_slider.Value = value

        controls_update
        return true
    end
End Function

```

### **Model\_window Control yaw\_slider:**

```

Sub ValueChanged()
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Yaw((val(yaw_edit.text) - (me.Value / 100.0)) * 3.1415926536)
    next

    controls_update
End Sub

```

### **Model\_window Control yaw\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim old_val as integer = yaw_slider.Value/100.0
        dim value as single = val(me.Text) * 100.0
        //psuedo modulo for dealing with single
        while value > 200.0
            value = value - 200.0
        wend
        while value < -200.0
            value = value + 200.0
        wend
        for i as integer = 0 to ubound(display_meshes)
            display_meshes(i).yaw((old_val - (value / 100.0)) * 3.1415926536)
        next
    end
End Function

```

```

yaw_slider.Value = value

controls_update
return true
end

```

End Function

### **Model\_window Control dcube\_check:**

```

Sub Action()
    if me.Value = true then
        Rb3DSpace1.DebugCube = true
        Rb3DSpace1.Update
    else
        Rb3DSpace1.DebugCube = false
        Rb3DSpace1.Update
    end
End Sub

```

End Sub

### **Model\_window Control LOD\_popup:**

```

Sub Change()
    dim index as integer = me.ListIndex

    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)

    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    model_data_list.DeleteAllRows
    meshes_list.DeleteAllRows
    dim used_meshes(-1) as integer

    for i as integer = 0 to ubound(Model1.model_chunk)
        dim mesh_to_use as integer
        select case index
            //the use of links(0) is to use the most basic of any of the model variants
            case 0
                //superhigh
                mesh_to_use = Model1.model_chunk(i).links(0).super_high
            case 1
                //high
                mesh_to_use = Model1.model_chunk(i).links(0).high
            case 2
                //med
                mesh_to_use = Model1.model_chunk(i).links(0).medium
            case 3
                //low
                mesh_to_use = Model1.model_chunk(i).links(0).low
            case 4
                //superlow

```

```

    mesh_to_use = Model1.model_chunk(i).links(0).super_low
end select
used_meshes.Append(mesh_to_use)

for j as integer = 0 to ubound(Model1.meshes(mesh_to_use).submodels)

    display_meshes.Append(Model1.meshes(mesh_to_use).submodels(j))

next

//show all possible versions of the displayed meshes
//if ubound(Model1.model_chunk(i).links) > 0 then
for j as integer = 0 to ubound(model1.model_chunk(i).links)
    model_data_list.addrow(model1.model_chunk(i).links(j).name)
    //only for the initial changes
    if j = 0 then
        model_data_list.cell(model_data_list.LastIndex, 1) = "Yes"
    else
        model_data_list.cell(model_data_list.LastIndex, 1) = "No"
    end
    model_data_list.cell(model_data_list.LastIndex, 2) = str(i)
    model_data_list.cell(model_data_list.LastIndex, 3) = str(j)
next
//end

next

//show the entire list of meshes and mark the displayed ones
for i as integer = 0 to ubound(model1.model_chunk)
    for j as integer = 0 to ubound(model1.model_chunk(i).links)
        dim used as boolean = false
        //do the super highs
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super High")
        if str(model1.model_chunk(i).links(j).super_high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_meshes)
            if used_meshes(k) = model1.model_chunk(i).links(j).super_high then
                used = true
                k = ubound(used_meshes)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end

        //do the high
        used = false
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " High")
    next
next

```

```

if str(model1.model_chunk(i).links(j).high).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).high)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).high)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).high then
        used = true
        k = ubound(used_mesches)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the medium
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Medium")
if str(model1.model_chunk(i).links(j).medium).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).medium)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).medium)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).medium then
        used = true
        k = ubound(used_mesches)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the low
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Low")
if str(model1.model_chunk(i).links(j).low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).low then
        used = true
        k = ubound(used_mesches)
    end
next

```

```

if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the super low
used=false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super Low")
if str(model1.model_chunk(i).links(j).super_low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_meshes)
    if used_meshes(k) = model1.model_chunk(i).links(j).super_low then
        used = true
        k = ubound(used_meshes)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

next
next

//show the cut off distance
//possibly make it editable
cut_off_field.text = str(model1.header.Lod_cutoffs(index))

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next

Rb3DSpace1.Update
End Sub

```

### **Model\_window Control skin\_push:**

```

Sub Action()
    dim f as folderitem = GetOpenFolderItem("")
    if f = nil then return

    dim temp as Picture = f.OpenAsPicture
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Material.Texture = temp
    next
    Rb3DSpace1.Update
End Sub

```

### **Model\_window Control model\_data\_list:**

```

Sub DoubleClick()
    Dim row as Integer
    row= Me.RowFromXY( System.MouseX - Me.Left - Self.Left, System.MouseY - Me.Top - Self.Top)

    //now get all the info about each of the models and which variant is used
    Dim models_links_indeces(-1) as integer
    for i as integer = 0 to ubound(model1.model_chunk)
        models_links_indeces.Append 0
    next
    for i as integer = 0 to me.ListCount-1
        if me.cell(i, 1) = "Yes" then
            models_links_indeces(val(me.cell(i, 2))) = val(me.cell(i, 3))
        end
    next

    models_links_indeces(val(me.cell(row, 2))) = val(me.cell(row, 3))

    dim index as integer = LOD_popup.ListIndex

    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)

    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    model_data_list.DeleteAllRows
    meshes_list.DeleteAllRows
    dim used_meshes(-1) as integer

    for i as integer = 0 to ubound(Model1.model_chunk)
        dim mesh_to_use as integer
        select case index
            //models_links_indeces is a way of keeping track of changes
            case 0
                //superhigh
                mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).super_high
            case 1
                //high
                mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).high
            case 2
                //med
                mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).medium
            case 3
                //low
                mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).low
            case 4
                //superlow
                mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).super_low
        end select
    next
end Sub

```



```

used_meshes.Append(mesh_to_use)

for j as integer = 0 to ubound(Model1.meshes(mesh_to_use).submodels)

    display_meshes.Append(Model1.meshes(mesh_to_use).submodels(j))

next

//show all possible versions of the displayed meshes
//if ubound(Model1.model_chunk(i).links) > 0 then
for j as integer = 0 to ubound(model1.model_chunk(i).links)
    model_data_list.addrow(model1.model_chunk(i).links(j).name)
    //only for the initial changes
    if j = models_links_indeces(i) then
        model_data_list.cell(model_data_list.LastIndex, 1) = "Yes"
    else
        model_data_list.cell(model_data_list.LastIndex, 1) = "No"
    end
    model_data_list.cell(model_data_list.LastIndex, 2) = str(i)
    model_data_list.cell(model_data_list.LastIndex, 3) = str(j)
next
//end

next

//show the entire list of meshes and mark the displayed ones
for i as integer = 0 to ubound(model1.model_chunk)
    for j as integer = 0 to ubound(model1.model_chunk(i).links)
        dim used as boolean = false
        //do the super highs
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super High")
        if str(model1.model_chunk(i).links(j).super_high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_meshes)
            if used_meshes(k) = model1.model_chunk(i).links(j).super_high then
                used = true
                k = ubound(used_meshes)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end

        //do the high
        used = false
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " High")
        if str(model1.model_chunk(i).links(j).high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).high)

```

```

else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).high)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_meshe)
    if used_meshe(k) = model1.model_chunk(i).links(j).high then
        used = true
        k = ubound(used_meshe)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the medium
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Medium")
if str(model1.model_chunk(i).links(j).medium).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).medium)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).medium)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_meshe)
    if used_meshe(k) = model1.model_chunk(i).links(j).medium then
        used = true
        k = ubound(used_meshe)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the low
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Low")
if str(model1.model_chunk(i).links(j).low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_meshe)
    if used_meshe(k) = model1.model_chunk(i).links(j).low then
        used = true
        k = ubound(used_meshe)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"

```

```

else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the super low
used=false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super Low")
if str(model1.model_chunk(i).links(j).super_low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_meshe)
    if used_meshe(k) = model1.model_chunk(i).links(j).super_low then
        used = true
        k = ubound(used_meshe)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

next
next

//show the cut off distance
//possibly make it editable
cut_off_field.text = str(model1.header.Lod_cutooffs(index))

for i as integer = 0 to ubound(display_meshe)
    Rb3DSpace1.Objects.Append display_meshe(i)
next

Rb3DSpace1.Update
End Sub

```

### **Model\_window Control attach\_list:**

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    dim i, j as integer

```

```

    i = val(me.Cell(row, 1))
    j = val(me.Cell(row, 2))

```

```

    show_attachment_point(i, j)
End Function

```

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim ok as boolean = false
    select case asc(key)
    case 31

```

```

        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end
end

if ok then
    dim i, j as integer

    i = val(me.Cell(row, 1))
    j = val(me.Cell(row, 2))

    show_attachment_point(i, j)
end
End Function

```

### **Model\_window Control sub\_model\_popup:**

```

Sub Change()
    //fill the listbox with each submodel

    sub_model_list.DeleteAllRows

    //need to develop code that will generate a given submodel mesh
    for i as integer = 0 to ubound(model1.meshes(me.ListIndex).submodels)
        if i < 9 then
            sub_model_list.AddRow("Submodel: 0" + str(i+1))
        else
            sub_model_list.AddRow("Submodel: " + str(i+1))
        end
    next

    sub_model_list.Selected(0) = true
    show_sub_model(me.ListIndex, 0)
End Sub

```

### **Model\_window Control sub\_model\_list:**

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    show_sub_model(sub_model_popup.ListIndex, row)
End Function

```

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    end
end

```

```

case 30
    if me.ListIndex - 1 > -1 then
        row = me.ListIndex - 1
        ok = true
    end
end

if ok then
    show_sub_model(sub_model_popup.ListIndex, row)
end

```

End Function

### **Model\_window Control bones\_list:**

Function KeyDown(Key As String) As Boolean

```

dim row as integer
dim ok as boolean = false
select case asc(key)
case 31
    if me.ListIndex + 1 < me.listcount then
        row = me.ListIndex + 1
        ok = true
    end
case 30
    if me.ListIndex - 1 > -1 then
        row = me.ListIndex - 1
        ok = true
    end
end

if ok then
    show_selected_bone(val(me.Cell(row, 0)))
end

```

End Function

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean

```

    show_selected_bone(val(me.Cell(row, 0)))

```

End Function

End Class

## **Class Mod2\_control**

Inherits ContainerControl

### **Mod2\_control.init:**

Sub init(in\_f as folderitem, magic as integer, offset as integer, index\_offset as integer, vertex\_offset as integer, in\_class2 as string, in\_fullname as string)

```

    f = in_f
    class2 = in_class2
    fullname = in_fullname
    in_index_offset = index_offset

```

```

    in_magic = magic
    in_offset = offset
    in_vertex_offset = vertex_offset
End Sub
class2 As string

```

```

f As folderItem

```

```

fullname As string

```

```

in_index_offset As Integer

```

```

in_magic As Integer

```

```

in_offset As Integer

```

```

in_vertex_offset As Integer

```

### **Mod2\_control Control preview\_model:**

```

Sub Action()
    dim check_model as new model
    check_model.read(f, in_magic, in_offset, in_index_offset, in_vertex_offset)

    Model_Window.Title = class2 + " " + fullname
    Model_Window.init(check_model, f)
    Model_Window.Show
End Sub
End Class

```

## **Class Model\_control**

Inherits ContainerControl

### **Model\_control.Close:**

```

Sub Close()
    'Rb3DSpace1.Close
End Sub

```

### **Model\_control.LostFocus:**

```

Sub LostFocus()
    MsgBox("hello")
End Sub

```

### **Model\_control.Open:**

```

Sub Open()
    #if DebugBuild
        skin_push.Visible = true
        skin_push.Enabled = true
    #else
        skin_push.Visible = false
        skin_push.Enabled = false
    #endif

```

End Sub

### **Model\_control.menuitemclose:**

Function menuitemclose() As Boolean

    me.Visible = false

End Function

### **Model\_control.controls\_update:**

Sub controls\_update()

    dim x as double = Rb3DSpace1.Camera.Position.X

    dim y as double = Rb3DSpace1.Camera.Position.Y

    dim z as double = Rb3DSpace1.Camera.Position.Z

    //the if loops are what allow the slider to center once it has reached an end point

    x\_slider.Value=x\*10

    if x\_slider.Value = x\_slider.Maximum then

        x\_slider.Minimum = x\_slider.Minimum + 300

        x\_slider.Maximum = x\_slider.Maximum + 300

    end

    if x\_slider.Value = x\_slider.Minimum then

        x\_slider.Minimum = x\_slider.Minimum - 300

        x\_slider.Maximum = x\_slider.Maximum - 300

    end

    y\_slider.Value=y\*10

    if y\_slider.Value = y\_slider.Maximum then

        y\_slider.Minimum = y\_slider.Minimum + 300

        y\_slider.Maximum = y\_slider.Maximum + 300

    end

    if y\_slider.Value = y\_slider.Minimum then

        y\_slider.Minimum = y\_slider.Minimum - 300

        y\_slider.Maximum = y\_slider.Maximum - 300

    end

    z\_slider.Value=z\*10

    if z\_slider.Value = z\_slider.Maximum then

        z\_slider.Minimum = z\_slider.Minimum + 300

        z\_slider.Maximum = z\_slider.Maximum + 300

    end

    if z\_slider.Value = z\_slider.Minimum then

        z\_slider.Minimum = z\_slider.Minimum - 300

        z\_slider.Maximum = z\_slider.Maximum - 300

    end

    x\_edit.text = str(x)

    y\_edit.text = str(y)

    z\_edit.text = str(z)

    pitch\_edit.text = str(pitch\_slider.value / 100.0)

    yaw\_edit.text = str(yaw\_slider.value / 100.0)

    roll\_edit.text = str(roll\_slider.value / 100.0)

    Rb3DSpace1.Update

End Sub

### **Model\_control.hide:**

```
Sub hide()  
    Rb3DSpace1.Width = 1  
    Rb3DSpace1.Height = 1  
    self.Refresh(true)  
    self.UpdateNow  
    self.Refresh(true)
```

End Sub

### **Model\_control.init:**

```
Sub init(in_model as model, in_f as folderitem)  
    model1 = in_model  
    f = in_f  
  
    redim meshes(-1)  
    for i as integer = 0 to ubound(in_model.meshes)  
        meshes.Append in_model.meshes(i)  
    next  
  
    Rb3DSpace1.Wireframe = true  
    Rb3DSpace1.DebugCube = false  
    roll_slider.Enabled = True  
    yaw_slider.Enabled = True  
    pitch_slider.Enabled = True  
    x_slider.Value = 0  
    y_slider.Value = 0  
    z_slider.Value = 0  
    control_tabs.value = 0
```

End Sub

### **Model\_control.return\_box:**

```
Protected Function return_box(x as single, y as single, z as single) As trimesh  
    //so we do  
  
    //x + 0.001, y + 0.001, z + 0.001  
    //x + 0.001, y + 0.001, z - 0.001  
    //x + 0.001, y - 0.001, z + 0.001  
    //x + 0.001, y - 0.001, z - 0.001  
    //x - 0.001, y + 0.001, z + 0.001  
    //x - 0.001, y + 0.001, z - 0.001  
    //x - 0.001, y - 0.001, z + 0.001  
    //x - 0.001, y - 0.001, z - 0.001  
  
    dim return_tm as new trimesh  
  
    //so lets add all those points  
    return_tm.VertexCount = 8  
    'return_tm.VertexPositions.SetXYZ(0, x + 0.001, y + 0.001, z + 0.001)  
    'return_tm.VertexPositions.SetXYZ(1, x + 0.001, y + 0.001, z - 0.001)  
    'return_tm.VertexPositions.SetXYZ(2, x + 0.001, y - 0.001, z + 0.001)  
    'return_tm.VertexPositions.SetXYZ(3, x + 0.001, y - 0.001, z - 0.001)
```



```

'return_tm.VertexPositions.SetXYZ(4, x - 0.001, y + 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(5, x - 0.001, y + 0.001, z - 0.001)
'return_tm.VertexPositions.SetXYZ(6, x - 0.001, y - 0.001, z + 0.001)
'return_tm.VertexPositions.SetXYZ(7, x - 0.001, y - 0.001, z - 0.001)
return_tm.VertexPositions.SetXYZ(0, x + 0.01, y + 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(1, x + 0.01, y + 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(2, x + 0.01, y - 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(3, x + 0.01, y - 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(4, x - 0.01, y + 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(5, x - 0.01, y + 0.01, z - 0.01)
return_tm.VertexPositions.SetXYZ(6, x - 0.01, y - 0.01, z + 0.01)
return_tm.VertexPositions.SetXYZ(7, x - 0.01, y - 0.01, z - 0.01)

```

```

return_tm.HasVertexColors = true
return_tm.VertexColors.Item(0) = &cFFFAFA
return_tm.VertexColors.Item(1) = &cFFFAFA
return_tm.VertexColors.Item(2) = &cFFFAFA
return_tm.VertexColors.Item(3) = &cFFFAFA
return_tm.VertexColors.Item(4) = &cFFFAFA
return_tm.VertexColors.Item(5) = &cFFFAFA
return_tm.VertexColors.Item(6) = &cFFFAFA
return_tm.VertexColors.Item(7) = &cFFFAFA

```

```

return_tm.HasVertexNormals = true
return_tm.VertexNormals.SetXYZ(0, 1, 1, 1)
return_tm.VertexNormals.SetXYZ(1, 1, 1, -1)
return_tm.VertexNormals.SetXYZ(2, 1, -1, 1)
return_tm.VertexNormals.SetXYZ(3, 1, -1, -1)
return_tm.VertexNormals.SetXYZ(4, -1, 1, 1)
return_tm.VertexNormals.SetXYZ(5, -1, 1, -1)
return_tm.VertexNormals.SetXYZ(6, -1, -1, 1)
return_tm.VertexNormals.SetXYZ(7, -1, -1, -1)

```

```

//now figure out all of the triangles necessary for rendering the box
//face in YZ-plane at x=1
//0, 1, 2
//1, 2, 3
//face in XZ-plane at y = 1
//0, 1, 5
//4, 0, 5
//face in YZ-plane at x = -1
//4, 5, 7
//7, 6, 4
//face in XZ-plane at y = -1
//2, 3, 7
//7, 6, 2
//face in XY-plane at z = 1
//2, 0, 4
//4, 6, 2
//face in XY-plane at z = -1
//3, 1, 5
//5, 7, 3

```

```

return_tm.TriangleCount = 12

```

```

return_tm.Triangles.SetABC(0, 0, 1, 2)
return_tm.Triangles.SetABC(1, 1, 2, 3)
return_tm.Triangles.SetABC(2, 0, 1, 5)
return_tm.Triangles.SetABC(3, 4, 0, 5)
return_tm.Triangles.SetABC(4, 4, 5, 7)
return_tm.Triangles.SetABC(5, 7, 6, 4)
return_tm.Triangles.SetABC(6, 2, 3, 7)
return_tm.Triangles.SetABC(7, 7, 6, 2)
return_tm.Triangles.SetABC(8, 2, 0, 4)
return_tm.Triangles.SetABC(9, 4, 6, 2)
return_tm.Triangles.SetABC(10, 3, 1, 5)
return_tm.Triangles.SetABC(11, 5, 7, 3)
return_tm.RenderBackFaces = true

```

```

return return_tm
End Function

```

### **Model\_control.setcolor:**

```

Private Function setcolor(in_color as color) As picture
    //thanks to G3David for providing this code
    //because I don't like Rb3DSpaces, RB all that much and hate bitmaps
    //oh how I hate you bitmaps
    //much hate...
    //so much hate...
    //btw, I haven't started on it yet so this is all premature anxiety but still...
    //...
    //hate

    dim p as new Picture(1,1,32)
    p.Graphics.ForeColor=in_Color
    p.Graphics.DrawRect (0,0,1,1)
    return p
End Function

```

### **Model\_control.show:**

```

Sub show()
    Rb3DSpace1.Width = me.Width - 142
    Rb3DSpace1.Height = me.Height - 280
    self.Refresh(true)
    self.UpdateNow
End Sub

```

### **Model\_control.show\_attachment\_point:**

```

Protected Sub show_attachment_point(i_index as integer, j_index as integer)
    wireframe_check.Value = false

    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    end while

```

```

wend

//so first display a single LoD (super high!)
for i as integer = 0 to ubound(Model1.model_chunk)
    for j as integer = 0 to ubound(Model1.meshes( Model1.model_chunk(i).links(0).super_high ).submodels)
        display_meshes.Append(Model1.meshes( Model1.model_chunk(i).links(0).super_high ).submodels(j))
    next
next

//now display the selected attachment point
display_meshes.Append( return_box( _
model1.attachment_points(i_index).points(j_index).translation(0) , _
model1.attachment_points(i_index).points(j_index).translation(1) , _
model1.attachment_points(i_index).points(j_index).translation(2) _
) )

attach_x_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(0))
attach_y_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(1))
attach_z_edit.text = str(model1.attachment_points(i_index).points(j_index).translation(2))

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next

Rb3DSpace1.Update
End Sub

```

### **Model\_control.show\_attachment\_points:**

```

Protected Sub show_attachment_points()
    //first it must delete all visible meshes
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    //create a list of all the possible attachment points
    //listbox 2nd column: attachment_point index, listbox 3rd column: attachment_point point index

    attach_list.DeleteAllRows
    for i as integer = 0 to ubound(model1.attachment_points)
        for j as integer = 0 to ubound(model1.attachment_points(i).points)
            if ubound(model1.attachment_points(i).points) > 0 then
                attach_list.AddRow(model1.attachment_points(i).name + ": " + str(1+j))
            else
                attach_list.AddRow(model1.attachment_points(i).name)
            end
            attach_list.Cell(attach_list.LastIndex, 1) = str(i)
            attach_list.Cell(attach_list.LastIndex, 2) = str(j)
        next
    next
end Sub

```

Rb3DSpace1.Update

```
//and initialize to the first element
//if there aren't any then ignore this
if attach_list.listcount > 0 then
    attach_list.Selected(0) = true
    show_attachment_point(0,0)
end
End Sub
```

### **Model\_control.show\_bone:**

Protected Function show\_bone(bone\_index as integer) As trimesh

```
//the other way to read all bones
//just add together all the parent bones coords

dim this_index as integer = bone_index
dim x as single = 0
dim y as single = 0
dim z as single = 0

while this_index > -1
    x = x + model1.bones(this_index).Translation(0)
    y = y + model1.bones(this_index).Translation(1)
    z = z + model1.bones(this_index).Translation(2)
    this_index = model1.bones(this_index).ParentNode
wend

dim tri as new Trimesh

tri = return_box( x, y, z )
tri.Material.Texture = setcolor(&cFFFFFF)
return tri
End Function
```

### **Model\_control.show\_bones:**

Protected Sub show\_bones()

```
wireframe_check.Value = false

yaw_slider.Value = 0
roll_slider.Value = 0
pitch_slider.Value = 0
controls_update
redim display_meshes(-1)
while Rb3DSpace1.Objects.Count > 0
    Rb3DSpace1.Objects.Remove(0)
wend

bones_list.DeleteAllRows
bones_list.ColumnAlignment(3) = 3
bones_list.ColumnAlignment(4) = 3
bones_list.ColumnAlignment(5) = 3
for i as integer = 0 to ubound(model1.bones)
    if i < 10 then
```

```

        bones_list.AddRow("0" + str(i))
    else
        bones_list.AddRow(str(i))
    end
    bones_list.Cell(bones_list.LastIndex, 1) = Model1.bones(i).name
    if model1.bones(i).ParentNode > -1 then
        bones_list.Cell(bones_list.LastIndex, 2) = Model1.bones( _
            model1.bones(i).ParentNode ).name
    else
        bones_list.Cell(bones_list.LastIndex, 2) = "None"
    end
    bones_list.Cell(bones_list.LastIndex, 3) = str(Model1.bones(i).Translation(0))
    bones_list.Cell(bones_list.LastIndex, 4) = str(Model1.bones(i).Translation(1))
    bones_list.Cell(bones_list.LastIndex, 5) = str(Model1.bones(i).Translation(2))
next

dim bones(-1) as Trimesh
for i as integer = 0 to UBound(model1.bones)
    bones.Append show_bone(i)
next

for i as integer = 0 to ubound(bones)
    display_meshes.Append(bones(i))
next
for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next
Rb3DSpace1.Update

bones_list.Selected(0) = true
show_selected_bone(0)
End Sub

```

### **Model\_control.show\_selected\_bone:**

```

Protected Sub show_selected_bone(actual_index as integer)
    //this method takes an index and alters the color of the correct bone

    //erase the current space
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend
    //white out all the current bones
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Material.Texture = setcolor(&cFFFFFF)
    next
    //for the alt method
    display_meshes(actual_index).Material.Texture = setcolor(&c00FF00)
    //rerender
    for i as integer = 0 to ubound(display_meshes)
        Rb3DSpace1.Objects.Append display_meshes(i)
    next
    Rb3DSpace1.Update
End Sub

```

**Model\_control.show\_sub\_model:**

```
Protected Sub show_sub_model(mesh_index as integer, submodel_index as integer)
    //I really don't know why I'm making this a method but I am
    //so screw you whoever is reading this and making fun of my coding
    //I went and had to research how everything works using only HMT and SparkEdit source codes
    //for the record, obj-c is an ugly language that deserves to be destroyed.
    //yes...destroyed. double past tense.
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update

    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    redim display_meshes(-1)
    display_meshes.append model1.meshes(mesh_index).submodels(submodel_index)

    for i as integer = 0 to ubound(display_meshes)
        Rb3DSpace1.Objects.Append display_meshes(i)
    next

    Rb3DSpace1.Update
End Sub
```

**Model\_control.show\_sub\_models:**

```
Protected Sub show_sub_models()
    yaw_slider.Value = 0
    roll_slider.Value = 0
    pitch_slider.Value = 0
    controls_update
    redim display_meshes(-1)
    while Rb3DSpace1.Objects.Count > 0
        Rb3DSpace1.Objects.Remove(0)
    wend

    Rb3DSpace1.Update

    sub_model_popup.DeleteAllRows
    sub_model_list.DeleteAllRows

    for i as integer = 0 to ubound(model1.meshes)
        //this is the first version. for a later version consider implementing
        if i < 10 then
            sub_model_popup.AddRow("Mesh #0" + str(i))
        else
            sub_model_popup.AddRow("Mesh #" + str(i))
        end
    next

    sub_model_popup.ListIndex = 0
    //the selected list index will then trigger another bit of code relating to the listbox
End Sub
```

display\_meshes() As trimesh

Protected f As folderitem

meshes() As LoDS\_submodels

model1 As model

### **Model\_control Control Rb3DSpace1:**

Sub Open()

```
if me.Objects = nil then
    errorbox("You need to install Quesa")
end
```

```
//set camera position (looking backwards)
'me.camera.position.Z = 1
'me.camera.position.X = 0
//me.camera.pitch 3.1415926536
//me.camera.roll 3.1415926536
```

```
//Rb3DSpace1.Update
```

End Sub

### **Model\_control Control control\_tabs:**

Sub Change()

```
select case me.value
```

```
case 0
```

```
//model data
LOD_popup.DeleteAllRows
```

```
LOD_popup.addrow("Super High")
LOD_popup.addrow("High")
LOD_popup.addrow("Medium")
LOD_popup.addrow("Low")
LOD_popup.addrow("Super Low")
```

```
LOD_popup.ListIndex = 0
```

```
case 1
```

```
//SubModels
show_sub_models
```

```
case 2
```

```
//bones
show_bones
```

```
case 3
```

```
//attachment points
show_attachment_points
```

```
end select
```

```
me.refresh
End Sub
```

### **Model\_control Control wireframe\_check:**

```
Sub Action()
    if me.Value = true then
        Rb3DSpace1.Wireframe = true
        Rb3DSpace1.Update
    else
        Rb3DSpace1.Wireframe = false
        Rb3DSpace1.Update
    end
End Sub
```

### **Model\_control Control x\_slider:**

```
Sub ValueChanged()
    Rb3DSpace1.Camera.Position.X = me.Value/10

    controls_update
End Sub
```

### **Model\_control Control x\_edit:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        x_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function
```

### **Model\_control Control y\_slider:**

```
Sub ValueChanged()
    Rb3DSpace1.Camera.Position.Y = me.Value/10

    controls_update
End Sub
```

### **Model\_control Control y\_edit:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        y_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function
```

### **Model\_control Control z\_slider:**

```
Sub ValueChanged()
    Rb3DSpace1.Camera.Position.Z = me.Value/10
```



```
controls_update
End Sub
```

### **Model\_control Control z\_edit:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim value as integer = val(me.Text)
        z_slider.Value = (value * 10)

        controls_update
        return true
    end
End Function
```

### **Model\_control Control pitch\_slider:**

```
Sub ValueChanged()
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Pitch((val(pitch_edit.text) - (me.Value / 100.0)) * 3.1415926536)
    next

    controls_update
End Sub
```

### **Model\_control Control pitch\_edit:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim old_val as integer = pitch_slider.Value/100.0
        dim value as single = val(me.Text) * 100.0
        //psuedo modulo for dealing with single
        while value > 200.0
            value = value - 200.0
        wend
        while value < -200.0
            value = value + 200.0
        wend
        for i as integer = 0 to ubound(display_meshes)
            display_meshes(i).Pitch((old_val - (value / 100.0)) * 3.1415926536)
        next
        pitch_slider.Value = value

        controls_update
        return true
    end
End Function
```

### **Model\_control Control roll\_slider:**

```
Sub ValueChanged()
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Roll((val(roll_edit.text) - (me.Value / 100.0)) * 3.1415926536)
    next
    controls_update
End Sub
```

**Model\_control Control roll\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim old_val as integer = roll_slider.Value/100.0
    dim value as single = val(me.Text) * 100.0
    //psuedo modulo for dealing with single
    while value > 200.0
        value = value - 200.0
    wend
    while value < -200.0
        value = value + 200.0
    wend
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Roll((old_val - (value / 100.0)) * 3.1415926536)
    next
    roll_slider.Value = value

    controls_update
    return true
end
```

End Function

**Model\_control Control yaw\_slider:**

Sub ValueChanged()

```
for i as integer = 0 to ubound(display_meshes)
    display_meshes(i).Yaw((val(yaw_edit.text) - (me.Value / 100.0)) * 3.1415926536)
next

controls_update
```

End Sub

**Model\_control Control yaw\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim old_val as integer = yaw_slider.Value/100.0
    dim value as single = val(me.Text) * 100.0
    //psuedo modulo for dealing with single
    while value > 200.0
        value = value - 200.0
    wend
    while value < -200.0
        value = value + 200.0
    wend
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).yaw((old_val - (value / 100.0)) * 3.1415926536)
    next
    yaw_slider.Value = value

    controls_update
    return true
end
```

End Function

Model\_control Control LOD\_popup:

Sub Change()

dim index as integer = me.ListIndex

yaw\_slider.Value = 0

roll\_slider.Value = 0

pitch\_slider.Value = 0

controls\_update

redim display\_meshes(-1)

while Rb3DSpace1.Objects.Count > 0

    Rb3DSpace1.Objects.Remove(0)

wend

model\_data\_list.DeleteAllRows

meshes\_list.DeleteAllRows

dim used\_meshes(-1) as integer

for i as integer = 0 to ubound(Model1.model\_chunk)

    dim mesh\_to\_use as integer

    select case index

        //the use of links(0) is to use the most basic of any of the model variants

    case 0

        //superhigh

        mesh\_to\_use = Model1.model\_chunk(i).links(0).super\_high

    case 1

        //high

        mesh\_to\_use = Model1.model\_chunk(i).links(0).high

    case 2

        //med

        mesh\_to\_use = Model1.model\_chunk(i).links(0).medium

    case 3

        //low

        mesh\_to\_use = Model1.model\_chunk(i).links(0).low

    case 4

        //superlow

        mesh\_to\_use = Model1.model\_chunk(i).links(0).super\_low

end select

used\_meshes.Append(mesh\_to\_use)

for j as integer = 0 to ubound(Model1.meshes(mesh\_to\_use).submodels)

    display\_meshes.Append(Model1.meshes(mesh\_to\_use).submodels(j))

next

//show all possible versions of the displayed meshes

//if ubound(Model1.model\_chunk(i).links) > 0 then

for j as integer = 0 to ubound(model1.model\_chunk(i).links)

    model\_data\_list.addrow(model1.model\_chunk(i).links(j).name)

    //only for the initial changes

    if j = 0 then

        model\_data\_list.cell(model\_data\_list.LastIndex, 1) = "Yes"

```

else
    model_data_list.cell(model_data_list.LastIndex, 1) = "No"
end
model_data_list.cell(model_data_list.LastIndex, 2) = str(i)
model_data_list.cell(model_data_list.LastIndex, 3) = str(j)
next
//end

next

//show the entire list of meshes and mark the displayed ones
for i as integer = 0 to ubound(model1.model_chunk)
    for j as integer = 0 to ubound(model1.model_chunk(i).links)
        dim used as boolean = false
        //do the super highs
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super High")
        if str(model1.model_chunk(i).links(j).super_high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_mesches)
            if used_mesches(k) = model1.model_chunk(i).links(j).super_high then
                used = true
                k = ubound(used_mesches)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end

        //do the high
        used = false
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " High")
        if str(model1.model_chunk(i).links(j).high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_mesches)
            if used_mesches(k) = model1.model_chunk(i).links(j).high then
                used = true
                k = ubound(used_mesches)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end
    end
end

```

```

//do the medium
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Medium")
if str(model1.model_chunk(i).links(j).medium).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).medium)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).medium)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).medium then
        used = true
        k = ubound(used_mesches)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the low
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Low")
if str(model1.model_chunk(i).links(j).low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).low then
        used = true
        k = ubound(used_mesches)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the super low
used=false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super Low")
if str(model1.model_chunk(i).links(j).super_low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).super_low then

```

```

        used = true
        k = ubound(used_meshes)
    end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

next
next

//show the cut off distance
//possibly make it editable
cut_off_field.text = str(model1.header.Lod_cutoffs(index))

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next

```

```

Rb3DSpace1.Update
End Sub

```

### **Model\_control Control skin\_push:**

```

Sub Action()
    dim f as folderitem = GetOpenFolderItem("")
    if f = nil then return

    dim temp as Picture = f.OpenAsPicture
    for i as integer = 0 to ubound(display_meshes)
        display_meshes(i).Material.Texture = temp
    next
    Rb3DSpace1.Update
End Sub

```

### **Model\_control Control model\_data\_list:**

```

Sub DoubleClick()
    Dim row as Integer
    row= Me.RowFromXY( System.MouseX - Me.Left - Self.Left, System.MouseY - Me.Top - Self.Top)

    //now get all the info about each of the models and which variant is used
    Dim models_links_indeces(-1) as integer
    for i as integer = 0 to ubound(model1.model_chunk)
        models_links_indeces.Append 0
    next
    for i as integer = 0 to me.ListCount-1
        if me.cell(i, 1) = "Yes" then
            models_links_indeces(val(me.cell(i, 2))) = val(me.cell(i, 3))
        end
    next

    models_links_indeces(val(me.cell(row, 2))) = val(me.cell(row, 3))

```

```

dim index as integer = LOD_popup.ListIndex

yaw_slider.Value = 0
roll_slider.Value = 0
pitch_slider.Value = 0
controls_update
redim display_meshes(-1)

while Rb3DSpace1.Objects.Count > 0
    Rb3DSpace1.Objects.Remove(0)
wend

model_data_list.DeleteAllRows
meshes_list.DeleteAllRows
dim used_meshes(-1) as integer

for i as integer = 0 to ubound(Model1.model_chunk)
    dim mesh_to_use as integer
    select case index
        //models_links_indeces is a way of keeping track of changes
    case 0
        //superhigh
        mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).super_high
    case 1
        //high
        mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).high
    case 2
        //med
        mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).medium
    case 3
        //low
        mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).low
    case 4
        //superlow
        mesh_to_use = Model1.model_chunk(i).links(models_links_indeces(i)).super_low
    end select
    used_meshes.Append(mesh_to_use)

    for j as integer = 0 to ubound(Model1.meshes(mesh_to_use).submodels)

        display_meshes.Append(Model1.meshes(mesh_to_use).submodels(j))

    next

    //show all possible versions of the displayed meshes
    //if ubound(Model1.model_chunk(i).links) > 0 then
    for j as integer = 0 to ubound(model1.model_chunk(i).links)
        model_data_list.addrow(model1.model_chunk(i).links(j).name)
        //only for the initial changes
        if j = models_links_indeces(i) then
            model_data_list.cell(model_data_list.LastIndex, 1) = "Yes"
        else
            model_data_list.cell(model_data_list.LastIndex, 1) = "No"
        end if
    next j
end for i

```

```

    end
    model_data_list.cell(model_data_list.LastIndex, 2) = str(i)
    model_data_list.cell(model_data_list.LastIndex, 3) = str(j)
next
//end

next

//show the entire list of meshes and mark the displayed ones
for i as integer = 0 to ubound(model1.model_chunk)
    for j as integer = 0 to ubound(model1.model_chunk(i).links)
        dim used as boolean = false
        //do the super highs
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super High")
        if str(model1.model_chunk(i).links(j).super_high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_meshes)
            if used_meshes(k) = model1.model_chunk(i).links(j).super_high then
                used = true
                k = ubound(used_meshes)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end

        //do the high
        used = false
        meshes_list.addrow(model1.model_chunk(i).links(j).name + " High")
        if str(model1.model_chunk(i).links(j).high).Len < 2 then
            meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).high)
        else
            meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).high)
        end
        meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
        for k as integer = 0 to ubound(used_meshes)
            if used_meshes(k) = model1.model_chunk(i).links(j).high then
                used = true
                k = ubound(used_meshes)
            end
        next
        if used then
            meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
        else
            meshes_list.cell(meshes_list.LastIndex, 2) = "No"
        end

        //do the medium

```



```

used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Medium")
if str(model1.model_chunk(i).links(j).medium).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).medium)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).medium)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).medium then
        used = true
        k = ubound(used_mesches)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the low
used = false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Low")
if str(model1.model_chunk(i).links(j).low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).low then
        used = true
        k = ubound(used_mesches)
    end
end
next
if used then
    meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
else
    meshes_list.cell(meshes_list.LastIndex, 2) = "No"
end

//do the super low
used=false
meshes_list.addrow(model1.model_chunk(i).links(j).name + " Super Low")
if str(model1.model_chunk(i).links(j).super_low).Len < 2 then
    meshes_list.cell(meshes_list.LastIndex, 1) = "0" + str(model1.model_chunk(i).links(j).super_low)
else
    meshes_list.cell(meshes_list.LastIndex, 1) = str(model1.model_chunk(i).links(j).super_low)
end
meshes_list.cellAlignment(meshes_list.LastIndex, 1) = 3
for k as integer = 0 to ubound(used_mesches)
    if used_mesches(k) = model1.model_chunk(i).links(j).super_low then
        used = true
        k = ubound(used_mesches)
    end
end

```

```

        end
    next
    if used then
        meshes_list.cell(meshes_list.LastIndex, 2) = "Yes"
    else
        meshes_list.cell(meshes_list.LastIndex, 2) = "No"
    end

    next
next

//show the cut off distance
//possibly make it editable
cut_off_field.text = str(model1.header.Lod_cutoffs(index))

for i as integer = 0 to ubound(display_meshes)
    Rb3DSpace1.Objects.Append display_meshes(i)
next

Rb3DSpace1.Update
End Sub

```

### **Model\_control Control attach\_list:**

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    dim i, j as integer

```

```

    i = val(me.Cell(row, 1))
    j = val(me.Cell(row, 2))

```

```

        show_attachment_point(i, j)
End Function

```

```

Function KeyDown(Key As String) As Boolean

```

```

    dim row as integer
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end
end

```

```

    if ok then
        dim i, j as integer

```

```

        i = val(me.Cell(row, 1))
        j = val(me.Cell(row, 2))

```

```

        show_attachment_point(i, j)
    end
End Function

```

### **Model\_control Control sub\_model\_popup:**

```

Sub Change()
    //fill the listbox with each submodel

    sub_model_list.DeleteAllRows

    //need to develop code that will generate a given submodel mesh
    for i as integer = 0 to ubound(model1.meshes(me.ListIndex).submodels)
        if i < 9 then
            sub_model_list.AddRow("Submodel: 0" + str(i+1))
        else
            sub_model_list.AddRow("Submodel: " + str(i+1))
        end
    next

    sub_model_list.Selected(0) = true
    show_sub_model(me.ListIndex, 0)
End Sub

```

### **Model\_control Control sub\_model\_list:**

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    show_sub_model(sub_model_popup.ListIndex, row)
End Function

```

```

Function KeyDown(Key As String) As Boolean
    dim row as integer
    dim ok as boolean = false
    select case asc(key)
    case 31
        if me.ListIndex + 1 < me.listcount then
            row = me.ListIndex + 1
            ok = true
        end
    case 30
        if me.ListIndex - 1 > -1 then
            row = me.ListIndex - 1
            ok = true
        end
    end

    if ok then
        show_sub_model(sub_model_popup.ListIndex, row)
    end
End Function

```

End Function

### **Model\_control Control bones\_list:**

```

Function KeyDown(Key As String) As Boolean

```

```

dim row as integer
dim ok as boolean = false
select case asc(key)
case 31
    if me.ListIndex + 1 < me.listcount then
        row = me.ListIndex + 1
        ok = true
    end
case 30
    if me.ListIndex - 1 > -1 then
        row = me.ListIndex - 1
        ok = true
    end
end

if ok then
    show_selected_bone(val(me.Cell(row, 0)))
end

```

End Function

```

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean
    show_selected_bone(val(me.Cell(row, 0)))
End Function
End Class

```

## **Class BSP\_display**

Inherits ContainerControl

### **BSP\_display.Open:**

```

Sub Open()
    '#if not DebugBuild
    'export_bsp.Visible = false
    '#endif
End Sub

```

### **BSP\_display.complement:**

```

Function complement(main_color as color) As color
    dim output as color

    //first method using hsv
    'dim h as double = main_color.Hue * 360
    'dim s as double = main_color.Saturation * 360
    'dim v as double = main_color.Value * 360

    //second method using CMY
    output = CMY(main_color.Red / 255.0, main_color.Green / 255.0, main_color.Blue / 255.0)

    return output
End Function

```

### **BSP\_display.init:**

```

Sub init(in_f as folderItem)
    f = in_f
End Sub

```

## **BSP\_display.load:**

```

Sub load(in_map as mapfile)
    //to whomever ends up reading this. yes I know this isn't completely mapped
    //right now I just want a functional bsp system

    //data loading
    map = in_map

    scenario = new Scenario_file
    scenario.init(map)

    dim br as BinaryStream = map.f.OpenAsBinaryFile
    br.LittleEndian = true
    br.position = scenario.header.StructBsp.offset - map.primarymagic

    dim BspStart() as UInt32
    dim BspSize() as UInt32
    dim BspMagic() as UInt32
    dim Zero1() as UInt32
    dim bsptag() as string
    dim namePtr() as uint32
    dim unknown2() as uint32
    dim tagid() as int32

    for i as integer = 1 to scenario.header.StructBsp.count
        BspStart.append br.readuint32
        BspSize.append br.ReadUInt32
        BspMagic.Append br.ReadUInt32
        Zero1.Append br.ReadUInt32
        bsptag.Append br.read(4)
        namePtr.append br.ReadUInt32 //can use this later
        unknown2.Append br.ReadUInt32
        tagid.Append br.ReadInt32
    next

    redim bsps(-1)
    for i as integer = 0 to scenario.header.StructBsp.count-1
        dim temp_bsp as new BSP_class
        temp_bsp.read(br, BspStart(i), BspMagic(i), mesh_color)
        bsps.Append temp_bsp
    next

    //gui initialization
    mesh_color_box.FillColor = mesh_color
    object_color = complement(mesh_color)
    change_ok = false
    bsp_popup.DeleteAllRows
    for i as integer = 1 to Scenario.header.StructBsp.count
        bsp_popup.AddRow("BSP #" + str(i))
    next

```

```

next
change_ok = true
bsp_popup.ListIndex = 0
change_ok = false

//normally would create the object list, but right now just setting up bsp manip
bsp_3d.Camera.Position.x = 0
bsp_3d.Camera.Position.y = 0
bsp_3d.Camera.Position.z = 10

dim pos as new Vector3D
pos.X = 0
pos.Y = 0
pos.Z = 100

dim light as new Light3D
light.Attenuation = 0
light.Position = pos
light.Brightness = 100
bsp_3d.AddLight(light)
bsp_3d.Update

//initialize the object selection
object_popup.DeleteAllRows
//add the other ones
object_popup.AddRow("Player Spawns")
object_popup.AddRow("Bipeds (SP)")
object_popup.AddRow("Scenery")
object_popup.AddRow("Vehicles")
object_popup.AddRow("Weapons")
object_popup.AddRow("Netgame Equipment")
object_popup.AddRow("Netgame Flags")
//select the first object from the list
change_ok = true
object_popup.ListIndex = 0
change_ok = false

change_ok = true

me.Refresh(true)
End Sub

```

### **BSP\_display.new\_box:**

```

Sub new_box()
    while bsp_3d.Objects.Count > 0
        bsp_3d.Objects.Remove(0)
    wend

    //readd the bsp stuff
    dim temp_ind as integer = bsp_popup.ListIndex
    for i as integer = 0 to UBound(bsps(temp_ind).bsp_tris)
        bsp_3d.Objects.Append bsps(temp_ind).bsp_tris(i)
    next

```

```

    object_box.Material.DiffuseColor = object_color
    //now add the new box
    bsp_3d.Objects.Append(object_box)
    box = true

    bsp_3d.Update
End Sub
box As boolean

bsps(-1) As bsp_class

change_ok As boolean

current_pitch As single

current_roll As single

current_yaw As single

f As folderItem

map As mapfile

mesh_color As color

object_box As trimesh

object_color As color

scenario As scenario_file

spawn_control_ref As containerControl

```

### **BSP\_display Control roll\_slider:**

```

Sub ValueChanged()
    if change_ok then
        bsp_3d.Camera.Roll((current_roll - (me.Value / 100.0))* 3.1415926536)
        current_roll = (me.Value / 100.0)
        bsp_3d.Update
    end
End Sub

```

### **BSP\_display Control x\_slider:**

```

Sub ValueChanged()
    if change_ok then
        bsp_3d.Camera.Position.x = me.Value
        bsp_3d.Light(0).Position.x = me.Value

        bsp_3d.Update

        if me.Value = me.Maximum then
            me.Enabled = false
            me.Maximum = me.Maximum + 100
        end if
    end if
End Sub

```

```

    me.Minimum = me.Minimum + 100
    me.Enabled = true
end

```

```

if me.Value = me.Minimum then
    me.Enabled = false
    me.Maximum = me.Maximum - 100
    me.Minimum = me.Minimum - 100
    me.Enabled = true
end

```

```

end

```

```

End Sub

```

### **BSP\_display Control y\_slider:**

```

Sub ValueChanged()

```

```

    if change_ok then

```

```

        bsp_3d.Camera.Position.y = me.Value
        bsp_3d.Light(0).Position.y = me.Value

```

```

        bsp_3d.Update

```

```

    if me.Value = me.Maximum then
        me.Enabled = false
        me.Maximum = me.Maximum + 100
        me.Minimum = me.Minimum + 100
        me.Enabled = true
    end

```

```

    if me.Value = me.Minimum then
        me.Enabled = false
        me.Maximum = me.Maximum - 100
        me.Minimum = me.Minimum - 100
        me.Enabled = true
    end

```

```

end

```

```

end

```

```

End Sub

```

### **BSP\_display Control bsp\_popup:**

```

Sub Change()

```

```

    if change_ok then

```

```

        //delete all existing items

```

```

        while bsp_3d.Objects.Count > 0

```

```

            bsp_3d.Objects.Remove(0)

```

```

        wend

```

```

        //add all of the meshes associated with a given bsp

```

```

        for i as integer = 0 to UBound(bsps(me.ListIndex).bsp_tris)

```

```

            bsp_3d.Objects.Append bsps(me.ListIndex).bsp_tris(i)

```

```

        next

```

```

        if box then

```

```

            bsp_3d.Objects.Append(object_box)

```

```

        end

```

```

        bsp_3d.Update

```

```

    end

```

```

End Sub

```



### **BSP\_display Control z\_slider:**

```
Sub ValueChanged()  
    if change_ok then  
        bsp_3d.Camera.Position.z = me.Value  
        bsp_3d.Light(0).Position.z = me.Value  
  
        bsp_3d.Update  
  
        if me.Value = me.Maximum then  
            me.Enabled = false  
            me.Maximum = me.Maximum + 100  
            me.Minimum = me.Minimum + 100  
            me.Enabled = true  
        end  
  
        if me.Value = me.Minimum then  
            me.Enabled = false  
            me.Maximum = me.Maximum - 100  
            me.Minimum = me.Minimum - 100  
            me.Enabled = true  
        end  
    end  
End Sub
```

### **BSP\_display Control pitch\_slider:**

```
Sub ValueChanged()  
    if change_ok then  
        bsp_3d.Camera.Pitch((current_pitch - (me.Value / 100.0))* 3.1415926536)  
        current_pitch = (me.Value / 100.0)  
        bsp_3d.Update  
    end  
End Sub
```

### **BSP\_display Control yaw\_slider:**

```
Sub ValueChanged()  
    if change_ok then  
        bsp_3d.Camera.yaw((current_yaw - (me.Value / 100.0))* 3.1415926536)  
        current_yaw = (me.Value / 100.0)  
        bsp_3d.Update  
    end  
End Sub
```

### **BSP\_display Control object\_popup:**

```
Sub Change()  
    dim selected_string as string = me.List(me.ListIndex)  
  
    if change_ok then  
  
        if spawn_control_ref <> nil then  
            try  
                spawn_control_ref.Close  
            catch NilObjectException
```

```

    end try
end

select case selected_string

case "Player Spawns"
    //make sure there's at least one spawn point
    if UBound(scenario.player_spawn) > -1 then
        dim spawn_con as new Player_Spawn_Control
        spawn_con.init(scenario, map.f, map.primarymagic, self)
        spawn_con.EmbedWithin(object_edit_box,0,10)
        spawn_control_ref = spawn_con
    end

case "Bipeds (SP)"
    //make sure there's at least one spawn point
    if UBound(scenario.biped) > -1 then
        dim spawn_con as new Biped_Control
        spawn_con.init(scenario, map.f, map.primarymagic, self)
        spawn_con.EmbedWithin(object_edit_box,0,10)
        spawn_control_ref = spawn_con
    end

case "Scenery"
    //make sure there's at least one spawn point
    if UBound(scenario.scenery_spawn) > -1 then
        dim spawn_con as new Scenery_Spawn_Control
        spawn_con.init(scenario, map.f, map.primarymagic, self)
        spawn_con.EmbedWithin(object_edit_box,0,10)
        spawn_control_ref = spawn_con
    end

case "Vehicles"
    //make sure there's at least one spawn point
    if UBound(scenario.vehicle_spawn) > -1 then
        dim spawn_con as new Vehi_Spawn_Control
        spawn_con.init(scenario, map.f, map.primarymagic, self)
        spawn_con.EmbedWithin(object_edit_box,0,10)
        spawn_control_ref = spawn_con
    end

case "Weapons"
    //make sure there's at least one spawn point
    if UBound(scenario.weapon_spawn) > -1 then
        dim spawn_con as new Weapon_Spawn_Control
        spawn_con.init(scenario, map.f, map.primarymagic, self)
        spawn_con.EmbedWithin(object_edit_box,0,10)
        spawn_control_ref = spawn_con
    end

case "Netgame Equipment"
    //make sure there's at least one spawn point
    if UBound(scenario.MP_equip) > -1 then
        dim spawn_con as new NetEquip_Spawn_Control

```

```

spawn_con.init(scenario, map.f, map.primarymagic, self)
spawn_con.EmbedWithin(object_edit_box,0,10)
spawn_control_ref = spawn_con
end

```

```

case "Netgame Flags"
//make sure there's at least one spawn point
if UBound(scenario.MP_flag) > -1 then
dim spawn_con as new NetFlag_Spawn_Control
spawn_con.init(scenario, map.f, map.primarymagic, self)
spawn_con.EmbedWithin(object_edit_box,0,10)
spawn_control_ref = spawn_con
end
end select

```

```

end
End Sub

```

### **BSP\_display Control mesh\_color\_push:**

```

Sub Action()
dim temp_color as color = mesh_color
if SelectColor(temp_color, "Select Mesh Color") then
//change the color of every mesh in the bsps array
for i as integer = 0 to UBound(bsps)
for j as integer = 0 to UBound(bsps(i).bsp_tris)
bsps(i).bsp_tris(j).Material.DiffuseColor = temp_color
next
next
//delete every current mesh
while bsp_3d.Objects.Count > 0
bsp_3d.Objects.Remove(0)
wend
dim temp_ind as integer = bsp_popup.ListIndex
for i as integer = 0 to UBound(bsps(temp_ind).bsp_tris)
bsp_3d.Objects.Append bsps(temp_ind).bsp_tris(i)
next

mesh_color = temp_color
mesh_color_box.FillColor = mesh_color

//update the color of the item box
object_color = complement(mesh_color)
object_box.Material.DiffuseColor = object_color
bsp_3d.Objects.Append(object_box)

bsp_3d.Update
end
End Sub

```

### **BSP\_display Control export\_bsp:**

```

Sub Action()
dim index as integer = bsp_popup.ListIndex
dim f as FolderItem = GetSaveFolderItem("", "BSP.obj")
if f = nil then return

```

```

if f.exists then
    f.Delete
end

```

```

dim tw as TextOutputStream = f.CreateTextFile
for i as integer = 0 to UBound(bsps(index).submesh_headers)
    for j as integer = 0 to bsp(index).submesh_headers(i).vertexCount1-1
        tw.WriteLine("v " + str(bsps(index).submesh_headers(i).x(j)) + " " _
            + str(bsps(index).submesh_headers(i).y(j)) + " " + _
            str(bsps(index).submesh_headers(i).z(j)))
    next
    for j as integer = 0 to bsp(index).submesh_headers(i).vertexCount1-1
        tw.WriteLine("vn " + str(bsps(index).submesh_headers(i).normals_x(j)) + " " _
            + str(bsps(index).submesh_headers(i).normals_y(j)) + " " + _
            str(bsps(index).submesh_headers(i).normals_z(j)))
    next
    for j as integer = 0 to bsp(index).submesh_headers(i).vertIndexCount - 1
        dim tri_ind as integer = bsp(index).submesh_headers(i).vertIndexOffset + j
        //face variant 1
        tw.WriteLine("f " + str(-1 * (bsp(index).submesh_headers(i).vertexCount1 -
            bsp(index).subTri_Ind_1(tri_ind))) _
            + "/" + str(-1 * (bsp(index).submesh_headers(i).vertexCount1 - bsp(index).subTri_Ind_1(tri_ind))) _
            + " " + str(-1 * (bsp(index).submesh_headers(i).vertexCount1 - bsp(index).subTri_Ind_2(tri_ind))) + _
            "/" + str(-1 * (bsp(index).submesh_headers(i).vertexCount1 - bsp(index).subTri_Ind_2(tri_ind))) + " " + _
            str(-1 * (bsp(index).submesh_headers(i).vertexCount1 - bsp(index).subTri_Ind_3(tri_ind))) + "/" + _
            str(-1 * (bsp(index).submesh_headers(i).vertexCount1 - bsp(index).subTri_Ind_3(tri_ind))) )

        'tw.WriteLine("f " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1))

        //face variant 2
        'tw.WriteLine("f " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _
        '+ "/" + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) + _
        "/" + str(bsps(index).subTri_Ind_3(tri_ind) + 1) + " " + _
        'str(bsps(index).subTri_Ind_2(tri_ind) + 1) + "/" + _
        'str(bsps(index).subTri_Ind_2(tri_ind) + 1))

        'tw.WriteLine("f " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_2(tri_ind) + 1))

        //face variant 3
        'tw.WriteLine("f " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
        '+ "/" + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) + _
        "/" + str(bsps(index).subTri_Ind_1(tri_ind) + 1) + " " + _
        'str(bsps(index).subTri_Ind_3(tri_ind) + 1) + "/" + _
        'str(bsps(index).subTri_Ind_3(tri_ind) + 1))

        'tw.WriteLine("f " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
        '+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _

```

```

'+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1))

//face variant 4
'tw.WriteLine("f " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
'+ "/" + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) + _
"/" + str(bsps(index).subTri_Ind_3(tri_ind) + 1) + " " + _
'str(bsps(index).subTri_Ind_1(tri_ind) + 1) + "/" + _
'str(bsps(index).subTri_Ind_1(tri_ind) + 1))

'tw.WriteLine("f " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1))

//face variant 5
'tw.WriteLine("f " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ "/" + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) + _
"/" + str(bsps(index).subTri_Ind_1(tri_ind) + 1) + " " + _
'str(bsps(index).subTri_Ind_2(tri_ind) + 1) + "/" + _
'str(bsps(index).subTri_Ind_2(tri_ind) + 1))

'tw.WriteLine("f " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_2(tri_ind) + 1))

//face variant 6
'tw.WriteLine("f " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ "/" + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) + _
"/" + str(bsps(index).subTri_Ind_2(tri_ind) + 1) + " " + _
'str(bsps(index).subTri_Ind_1(tri_ind) + 1) + "/" + _
'str(bsps(index).subTri_Ind_1(tri_ind) + 1))

'tw.WriteLine("f " + str(bsps(index).subTri_Ind_3(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_2(tri_ind) + 1) _
'+ " " + str(bsps(index).subTri_Ind_1(tri_ind) + 1))
next
next

```

MsgBox("BSP Extracted Successfully!")

End Sub

## **BSP\_display Control PushButton1:**

Sub Action()

dim m\_map as new mapfile

m\_map.f = f

if m\_map.load then

bsp\_3d.Enabled = true

load(m\_map)

mesh\_color\_push.Enabled = true

export\_bsp.Enabled = true

pitch\_slider.Enabled = true

roll\_slider.Enabled = true

```

yaw_slider.Enabled = true
x_slider.Enabled = true
y_slider.Enabled = true
z_slider.Enabled = true
object_popup.Enabled = true
bsp_popup.Enabled = true
else
    errorbox("Error: There was a problem loading the scenario file")
end
End Sub
End Class

```

## **Class Player\_Spawn\_Control**

Inherits ContainerControl

### **Player\_Spawn\_Control.init:**

```

Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenario = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    //consider listing only the types of the spawn points and then having another list of each one
    //too damn many
    for i as integer = 0 to UBound(Scenario.player_spawn)
        spawn_select.AddRow("Player Spawn #" + str(i+1))
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub

```

### **Player\_Spawn\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.player_spawn(spawn_ind).x, _
    scenario.player_spawn(spawn_ind).y, scenario.player_spawn(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box

```

End Sub

### **Player\_Spawn\_Control.update\_info:**

Sub update\_info()

```
team_index_edit.text = str(scenario.player_spawn(spawn_ind).team_index)
bsp_index_edit.text = str(scenario.player_spawn(spawn_ind).bsp_index)
```

```
x_edit.text = str(scenario.player_spawn(spawn_ind).x)
y_edit.text = str(scenario.player_spawn(spawn_ind).y)
z_edit.text = str(scenario.player_spawn(spawn_ind).z)
```

```
roll_edit.Text = str(scenario.player_spawn(spawn_ind).rot)
```

```
dim type_string as string
select case Scenario.player_spawn(spawn_ind).gametype_enum1
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select
```

```
type_text1.text = "Type1: " + type_string
```

```
select case Scenario.player_spawn(spawn_ind).gametype_enum2
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
```

```

case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

```

```

type_text2.text = "Type2: " + type_string

```

```

select case Scenario.player_spawn(spawn_ind).gametype_enum3
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

```

```

type_text3.text = "Type3: " + type_string

```

```

select case Scenario.player_spawn(spawn_ind).gametype_enum4
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"

```



```

case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

```

```

type_text4.text = "Type4: " + type_string
End Sub
box As trimesh

```

```
f As folderitem
```

```
magic As Integer
```

```
scenario As scenario_file
```

```
spawn_ind As Integer
```

```
target As bsp_display
```

### **Player\_Spawn\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **Player\_Spawn\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex
    new_box
    update_info
End Sub

```

### **Player\_Spawn\_Control Control x\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position
    bw.WriteSingle(scenario.player_spawn(spawn_ind).x - 1.0)

```

```

    bw.close
    scenario.player_spawn(spawn_ind).x = scenario.player_spawn(spawn_ind).x - 1.0
    update_info
    new_box
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position
    bw.WriteSingle(scenario.player_spawn(spawn_ind).x + 1.0)
    bw.close
    scenario.player_spawn(spawn_ind).x = scenario.player_spawn(spawn_ind).x + 1.0
    update_info
    new_box
End Sub

```

### **Player\_Spawn\_Control Control x\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.player_spawn(spawn_ind).position
        bw.WriteSingle(val(x_edit.text))
        bw.close
        scenario.player_spawn(spawn_ind).x = val(x_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **Player\_Spawn\_Control Control y\_updown:**

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position + 4
    bw.WriteSingle(scenario.player_spawn(spawn_ind).y + 1.0)
    bw.close
    scenario.player_spawn(spawn_ind).y = scenario.player_spawn(spawn_ind).y + 1.0
    update_info
    new_box
End Sub

```

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position + 4
    bw.WriteSingle(scenario.player_spawn(spawn_ind).y - 1.0)
    bw.close
    scenario.player_spawn(spawn_ind).y = scenario.player_spawn(spawn_ind).y - 1.0
    update_info
    new_box
End Sub

```

**Player\_Spawn\_Control Control y\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position + 4
    bw.WriteSingle(val(y_edit.text))
    bw.close
    scenario.player_spawn(spawn_ind).y = val(y_edit.text)
    update_info
    new_box
    return true
end
```

End Function

**Player\_Spawn\_Control Control z\_updown:**

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.player_spawn(spawn_ind).position + 8
bw.WriteSingle(scenario.player_spawn(spawn_ind).z - 1.0)
bw.close
scenario.player_spawn(spawn_ind).z = scenario.player_spawn(spawn_ind).z - 1.0
update_info
new_box
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.player_spawn(spawn_ind).position + 8
bw.WriteSingle(scenario.player_spawn(spawn_ind).z + 1.0)
bw.close
scenario.player_spawn(spawn_ind).z = scenario.player_spawn(spawn_ind).z + 1.0
update_info
new_box
```

End Sub

**Player\_Spawn\_Control Control z\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position + 8
    bw.WriteSingle(val(z_edit.text))
    bw.close
    scenario.player_spawn(spawn_ind).z = val(z_edit.text)
    update_info
    new_box
    return true
end
```

End Function

Player\_Spawn\_Control Control roll\_updown:

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.player_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.player_spawn(spawn_ind).rot - 1.0)
bw.close
scenario.player_spawn(spawn_ind).rot = scenario.player_spawn(spawn_ind).rot - 1.0
update_info
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.player_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.player_spawn(spawn_ind).rot + 1.0)
bw.close
scenario.player_spawn(spawn_ind).rot = scenario.player_spawn(spawn_ind).rot + 1.0
update_info
```

End Sub

**Player\_Spawn\_Control Control roll\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.player_spawn(spawn_ind).position + 12
    bw.WriteSingle(val(roll_edit.text))
    bw.close
    scenario.player_spawn(spawn_ind).rot = val(roll_edit.text)
    update_info
    return true
end
```

End Function

End Class

## **Class biped\_Control**

Inherits ContainerControl

**biped\_Control.init:**

Sub init(byref in\_scen as scenario\_file, in\_f as folderItem, in\_magic as integer, byref in\_target as bsp\_display)

```
scenario = in_scen
f = in_f
magic = in_magic
target = in_target
```

```
spawn_select.DeleteAllRows
```

```
//ok, now list all of the existing spawn points
for i as integer = 0 to UBound(Scenario.biped)
    spawn_select.AddRow(scenario.biped(i).name)
```

```

next
spawn_select.ListIndex = 0
//and now initialize the first selection
spawn_ind = 0
new_box
update_info
End Sub

```

### **biped\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.biped(spawn_ind).x, _
    scenario.biped(spawn_ind).y, scenario.biped(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box
End Sub

```

### **biped\_Control.update\_info:**

```

Sub update_info()
    //break
    dim name_index as integer = Scenario.biped(spawn_ind).name_index
    if name_index <> -1 then
        dim name_string as string = scenario.object_names(name_index).name
        name_static.text = name_string
    else
        name_static.Text = "Script name not defined"
    end

    body_vitality_edit.text = str(scenario.biped(spawn_ind).body_vitality)
    if scenario.biped(spawn_ind).dead then
        dead_text.Text = "Start Biped as Dead"
    else
        dead_text.Text = "Start Biped as Alive"
    end

    x_edit.text = str(scenario.biped(spawn_ind).x)
    y_edit.text = str(scenario.biped(spawn_ind).y)
    z_edit.text = str(scenario.biped(spawn_ind).z)

    roll_edit.Text = str(scenario.biped(spawn_ind).roll)
    pitch_edit.Text = str(scenario.biped(spawn_ind).pitch)
    yaw_edit.Text = str(scenario.biped(spawn_ind).yaw)

    if scenario.biped(spawn_ind).not_placed_auto then
        auto_placed_text.Text = "Auto Placed: No"
    else

```

```

        auto_placed_text.Text = "Auto Placed: Yes"
    end
    if scenario.biped(spawn_ind).not_placed_easy then
        easy_placed_text.Text = "Placed on Easy: No"
    else
        easy_placed_text.Text = "Placed on Easy: Yes"
    end
    if scenario.biped(spawn_ind).not_placed_normal then
        norm_placed_text.Text = "Placed on Normal: No"
    else
        norm_placed_text.Text = "Placed on Normal: Yes"
    end
    if scenario.biped(spawn_ind).not_placed_hard then
        hard_placed_text.Text = "Placed on Hard: No"
    else
        hard_placed_text.Text = "Placed on Hard: Yes"
    end
End Sub
box As trimesh

```

f As folderitem

magic As Integer

scenario As scenario\_file

spawn\_ind As Integer

target As bsp\_display

### **biped\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **biped\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex
    new_box
    update_info
End Sub

```

### **biped\_Control Control x\_updown:**

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 8
    bw.WriteSingle(scenario.biped(spawn_ind).x + 1.0)
    bw.close
    scenario.biped(spawn_ind).x = scenario.biped(spawn_ind).x + 1.0
    update_info
    new_box

```

End Sub

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 8
bw.WriteSingle(scenario.biped(spawn_ind).x - 1.0)
bw.close
scenario.biped(spawn_ind).x = scenario.biped(spawn_ind).x - 1.0
update_info
new_box
```

End Sub

### **biped\_Control Control x\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 8
    bw.WriteSingle(val(x_edit.text))
    bw.close
    scenario.biped(spawn_ind).x = val(x_edit.text)
    update_info
    new_box
    return true
end
```

End Function

### **biped\_Control Control y\_updown:**

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 12
bw.WriteSingle(scenario.biped(spawn_ind).y + 1.0)
bw.close
scenario.biped(spawn_ind).y = scenario.biped(spawn_ind).y + 1.0
update_info
new_box
```

End Sub

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 12
bw.WriteSingle(scenario.biped(spawn_ind).y - 1.0)
bw.close
scenario.biped(spawn_ind).y = scenario.biped(spawn_ind).y - 1.0
update_info
new_box
```

End Sub

### **biped\_Control Control y\_edit:**

Function KeyDown(Key As String) As Boolean

```

if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 12
    bw.WriteSingle(val(y_edit.text))
    bw.close
    scenario.biped(spawn_ind).y = val(y_edit.text)
    update_info
    new_box
    return true
end
End Function

```

### **biped\_Control Control z\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 16
    bw.WriteSingle(scenario.biped(spawn_ind).z - 1.0)
    bw.close
    scenario.biped(spawn_ind).z = scenario.biped(spawn_ind).z - 1.0
    update_info
    new_box
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 16
    bw.WriteSingle(scenario.biped(spawn_ind).z + 1.0)
    bw.close
    scenario.biped(spawn_ind).z = scenario.biped(spawn_ind).z + 1.0
    update_info
    new_box
End Sub

```

### **biped\_Control Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.biped(spawn_ind).position + 16
        bw.WriteSingle(val(z_edit.text))
        bw.close
        scenario.biped(spawn_ind).z = val(z_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **biped\_Control Control roll\_updown:**

```

Sub Down()

```



```

dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 28
bw.WriteSingle(scenario.biped(spawn_ind).roll - 1.0)
bw.close
scenario.biped(spawn_ind).roll = scenario.biped(spawn_ind).roll - 1.0
update_info
End Sub

```

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 28
bw.WriteSingle(scenario.biped(spawn_ind).roll + 1.0)
bw.close
scenario.biped(spawn_ind).roll = scenario.biped(spawn_ind).roll + 1.0
update_info
End Sub

```

### **biped\_Control Control roll\_edit:**

```

Function KeyDown(Key As String) As Boolean
if asc(key) = 3 or asc(key) = 13 then
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 28
bw.WriteSingle(val(roll_edit.text))
bw.close
scenario.biped(spawn_ind).roll = val(roll_edit.text)
update_info
return true
end
End Function

```

### **biped\_Control Control pitch\_updown:**

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 24
bw.WriteSingle(scenario.biped(spawn_ind).pitch - 1.0)
bw.close
scenario.biped(spawn_ind).pitch = scenario.biped(spawn_ind).pitch - 1.0
update_info
End Sub

```

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 24
bw.WriteSingle(scenario.biped(spawn_ind).pitch + 1.0)
bw.close
scenario.biped(spawn_ind).pitch = scenario.biped(spawn_ind).pitch + 1.0
update_info
End Sub

```

biped\_Control Control pitch\_edit:

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 24
    bw.WriteSingle(val(pitch_edit.text))
    bw.close
    scenario.biped(spawn_ind).pitch = val(pitch_edit.text)
    update_info
    return true
end
```

End Function

**biped\_Control Control yaw\_updown:**

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 20
bw.WriteSingle(scenario.biped(spawn_ind).yaw - 1.0)
bw.close
scenario.biped(spawn_ind).yaw = scenario.biped(spawn_ind).yaw - 1.0
update_info
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.biped(spawn_ind).position + 20
bw.WriteSingle(scenario.biped(spawn_ind).yaw + 1.0)
bw.close
scenario.biped(spawn_ind).yaw = scenario.biped(spawn_ind).yaw + 1.0
update_info
```

End Sub

**biped\_Control Control yaw\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.biped(spawn_ind).position + 20
    bw.WriteSingle(val(yaw_edit.text))
    bw.close
    scenario.biped(spawn_ind).yaw = val(yaw_edit.text)
    update_info
    return true
end
```

End Function

End Class

## **Class Scenery\_Spawn\_Control**

Inherits ContainerControl

### **Scenery\_Spawn\_Control.init:**

```
Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenario = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    for i as integer = 0 to UBound(Scenario.scenery_spawn)
        spawn_select.AddRow(scenario.scenery_spawn(i).name)
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub
```

### **Scenery\_Spawn\_Control.new\_box:**

```
Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.scenery_spawn(spawn_ind).x, _
    scenario.scenery_spawn(spawn_ind).y, scenario.scenery_spawn(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box
End Sub
```

### **Scenery\_Spawn\_Control.update\_info:**

```
Sub update_info()
    //break
    dim name_index as integer = Scenario.scenery_spawn(spawn_ind).name_index
    if name_index <> -1 then
        dim name_string as string = scenario.object_names(name_index).name
        name_static.text = name_string
    else
        name_static.Text = "Script name not defined"
    end

    x_edit.text = str(scenario.scenery_spawn(spawn_ind).x)
    y_edit.text = str(scenario.scenery_spawn(spawn_ind).y)
    z_edit.text = str(scenario.scenery_spawn(spawn_ind).z)
```

```

roll_edit.Text = str(scenario.scenery_spawn(spawn_ind).roll)
pitch_edit.Text = str(scenario.scenery_spawn(spawn_ind).pitch)
yaw_edit.Text = str(scenario.scenery_spawn(spawn_ind).yaw)

if scenario.scenery_spawn(spawn_ind).not_placed_auto then
    auto_placed_text.Text = "Auto Placed: No"
else
    auto_placed_text.Text = "Auto Placed: Yes"
end
if scenario.scenery_spawn(spawn_ind).not_placed_easy then
    easy_placed_text.Text = "Placed on Easy: No"
else
    easy_placed_text.Text = "Placed on Easy: Yes"
end
if scenario.scenery_spawn(spawn_ind).not_placed_normal then
    norm_placed_text.Text = "Placed on Normal: No"
else
    norm_placed_text.Text = "Placed on Normal: Yes"
end
if scenario.scenery_spawn(spawn_ind).not_placed_hard then
    hard_placed_text.Text = "Placed on Hard: No"
else
    hard_placed_text.Text = "Placed on Hard: Yes"
end
End Sub
box As trimesh

f As folderitem

magic As Integer

scenario As scenario_file

spawn_ind As Integer

target As bsp_display

```

### **Scenery\_Spawn\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **Scenery\_Spawn\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex
    new_box
    update_info
End Sub

```

### **Scenery\_Spawn\_Control Control x\_updown:**

```

Sub Up()

```

```

dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.scenery_spawn(spawn_ind).position + 8
bw.WriteSingle(scenario.scenery_spawn(spawn_ind).x + 1.0)
bw.close
scenario.scenery_spawn(spawn_ind).x = scenario.scenery_spawn(spawn_ind).x + 1.0
update_info
new_box
End Sub

```

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.scenery_spawn(spawn_ind).position + 8
bw.WriteSingle(scenario.scenery_spawn(spawn_ind).x - 1.0)
bw.close
scenario.scenery_spawn(spawn_ind).x = scenario.scenery_spawn(spawn_ind).x - 1.0
update_info
new_box
End Sub

```

### **Scenery\_Spawn\_Control Control x\_edit:**

```

Function KeyDown(Key As String) As Boolean
if asc(key) = 3 or asc(key) = 13 then
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.scenery_spawn(spawn_ind).position + 8
bw.WriteSingle(val(x_edit.text))
bw.close
scenario.scenery_spawn(spawn_ind).x = val(x_edit.text)
update_info
new_box
return true
end
End Function

```

### **Scenery\_Spawn\_Control Control y\_updown:**

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.scenery_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.scenery_spawn(spawn_ind).y + 1.0)
bw.close
scenario.scenery_spawn(spawn_ind).y = scenario.scenery_spawn(spawn_ind).y + 1.0
update_info
new_box
End Sub

```

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.scenery_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.scenery_spawn(spawn_ind).y - 1.0)
bw.close

```

```

scenario.scenery_spawn(spawn_ind).y = scenario.scenery_spawn(spawn_ind).y - 1.0
update_info
new_box
End Sub

```

### **Scenery\_Spawn\_Control Control y\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.scenery_spawn(spawn_ind).position + 12
        bw.WriteSingle(val(y_edit.text))
        bw.close
        scenario.scenery_spawn(spawn_ind).y = val(y_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **Scenery\_Spawn\_Control Control z\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 16
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).z - 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).z = scenario.scenery_spawn(spawn_ind).z - 1.0
    update_info
    new_box
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 16
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).z + 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).z = scenario.scenery_spawn(spawn_ind).z + 1.0
    update_info
    new_box
End Sub

```

### **Scenery\_Spawn\_Control Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.scenery_spawn(spawn_ind).position + 16
        bw.WriteSingle(val(z_edit.text))
        bw.close
        scenario.scenery_spawn(spawn_ind).z = val(z_edit.text)
        update_info
    end
End Function

```

```

        new_box
    return true
end

```

End Function

### **Scenery\_Spawn\_Control Control roll\_updown:**

Sub Down()

```

    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).roll - 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).roll = scenario.scenery_spawn(spawn_ind).roll - 1.0
    update_info

```

End Sub

Sub Up()

```

    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).roll + 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).roll = scenario.scenery_spawn(spawn_ind).roll + 1.0
    update_info

```

End Sub

### **Scenery\_Spawn\_Control Control roll\_edit:**

Function KeyDown(Key As String) As Boolean

```

    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.scenery_spawn(spawn_ind).position + 28
        bw.WriteSingle(val(roll_edit.text))
        bw.close
        scenario.scenery_spawn(spawn_ind).roll = val(roll_edit.text)
        update_info
        return true
    end

```

End Function

### **Scenery\_Spawn\_Control Control pitch\_updown:**

Sub Down()

```

    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 24
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).pitch - 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).pitch = scenario.scenery_spawn(spawn_ind).pitch - 1.0
    update_info

```

End Sub

Sub Up()

```

    dim bw as BinaryStream = f.OpenAsBinaryFile(true)

```

```

    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 24
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).pitch + 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).pitch = scenario.scenery_spawn(spawn_ind).pitch + 1.0
    update_info
End Sub

```

### **Scenery\_Spawn\_Control Control pitch\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.scenery_spawn(spawn_ind).position + 24
        bw.WriteSingle(val(pitch_edit.text))
        bw.close
        scenario.scenery_spawn(spawn_ind).pitch = val(pitch_edit.text)
        update_info
        return true
    end
End Function

```

### **Scenery\_Spawn\_Control Control yaw\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 20
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).yaw - 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).yaw = scenario.scenery_spawn(spawn_ind).yaw - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.scenery_spawn(spawn_ind).position + 20
    bw.WriteSingle(scenario.scenery_spawn(spawn_ind).yaw + 1.0)
    bw.close
    scenario.scenery_spawn(spawn_ind).yaw = scenario.scenery_spawn(spawn_ind).yaw + 1.0
    update_info
End Sub

```

### **Scenery\_Spawn\_Control Control yaw\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.scenery_spawn(spawn_ind).position + 20
        bw.WriteSingle(val(yaw_edit.text))
        bw.close
        scenario.scenery_spawn(spawn_ind).yaw = val(yaw_edit.text)
        update_info
    end
End Function

```



```

        return true
    end
End Function
End Class

```

## **Class Vehi Spawn Control**

Inherits ContainerControl

### **Vehi\_Spawn\_Control.init:**

```

Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenario = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    for i as integer = 0 to UBound(Scenario.vehicle_spawn)
        spawn_select.AddRow(scenario.vehicle_spawn(i).name)
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub

```

### **Vehi\_Spawn\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.vehicle_spawn(spawn_ind).x, _
    scenario.vehicle_spawn(spawn_ind).y, scenario.vehicle_spawn(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box
End Sub

```

### **Vehi\_Spawn\_Control.update\_info:**

```

Sub update_info()
    //break
    dim name_index as integer = Scenario.vehicle_spawn(spawn_ind).name_index
    if name_index <> -1 then
        dim name_string as string = scenario.object_names(name_index).name
        name_static.text = name_string
    end if
End Sub

```

```

else
    name_static.Text = "Script name not defined"
end

body_vitality_edit.text = str(scenario.vehicle_spawn(spawn_ind).body_vitality)
team_index_edit.text = str(scenario.vehicle_spawn(spawn_ind).team_index)

x_edit.text = str(scenario.vehicle_spawn(spawn_ind).x)
y_edit.text = str(scenario.vehicle_spawn(spawn_ind).y)
z_edit.text = str(scenario.vehicle_spawn(spawn_ind).z)

roll_edit.Text = str(scenario.vehicle_spawn(spawn_ind).roll)
pitch_edit.Text = str(scenario.vehicle_spawn(spawn_ind).pitch)
yaw_edit.Text = str(scenario.vehicle_spawn(spawn_ind).yaw)

if scenario.vehicle_spawn(spawn_ind).not_placed_auto then
    auto_placed_text.Text = "Auto Placed: No"
else
    auto_placed_text.Text = "Auto Placed: Yes"
end
if scenario.vehicle_spawn(spawn_ind).not_placed_easy then
    easy_placed_text.Text = "Placed on Easy: No"
else
    easy_placed_text.Text = "Placed on Easy: Yes"
end
if scenario.vehicle_spawn(spawn_ind).not_placed_normal then
    norm_placed_text.Text = "Placed on Normal: No"
else
    norm_placed_text.Text = "Placed on Normal: Yes"
end
if scenario.vehicle_spawn(spawn_ind).not_placed_hard then
    hard_placed_text.Text = "Placed on Hard: No"
else
    hard_placed_text.Text = "Placed on Hard: Yes"
end

MP_flag_list.DeleteAllRows
MP_flag_list.AddRow("Slayer Default:")
if scenario.vehicle_spawn(spawn_ind).MP_slayer_default then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("CTF Default:")
if scenario.vehicle_spawn(spawn_ind).MP_ctf_default then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("King Default:")
if scenario.vehicle_spawn(spawn_ind).MP_king_default then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end

```

```

end
MP_flag_list.AddRow("Oddball Default:")
if scenario.vehicle_spawn(spawn_ind).MP_oddball_default then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("Slayer Allowed:")
if scenario.vehicle_spawn(spawn_ind).MP_slayer_ok then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("CTF Allowed:")
if scenario.vehicle_spawn(spawn_ind).MP_ctf_ok then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("King Allowed:")
if scenario.vehicle_spawn(spawn_ind).MP_king_ok then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
MP_flag_list.AddRow("Oddball Allowed:")
if scenario.vehicle_spawn(spawn_ind).MP_oddball_ok then
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[X]"
else
    MP_flag_list.Cell(MP_flag_list.LastIndex, 1) = "[ ]"
end
End Sub
box As trimesh

```

f As folderitem

magic As Integer

scenario As scenario\_file

spawn\_ind As Integer

target As bsp\_display

### **Vehi\_Spawn\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **Vehi\_Spawn\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex
    new_box

```

```
    update_info
```

```
End Sub
```

### **Vehi\_Spawn\_Control Control x\_updown:**

```
Sub Up()
```

```
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
```

```
    bw.LittleEndian = true
```

```
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 8
```

```
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).x + 1.0)
```

```
    bw.close
```

```
    scenario.vehicle_spawn(spawn_ind).x = scenario.vehicle_spawn(spawn_ind).x + 1.0
```

```
    update_info
```

```
    new_box
```

```
End Sub
```

```
Sub Down()
```

```
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
```

```
    bw.LittleEndian = true
```

```
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 8
```

```
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).x - 1.0)
```

```
    bw.close
```

```
    scenario.vehicle_spawn(spawn_ind).x = scenario.vehicle_spawn(spawn_ind).x - 1.0
```

```
    update_info
```

```
    new_box
```

```
End Sub
```

### **Vehi\_Spawn\_Control Control x\_edit:**

```
Function KeyDown(Key As String) As Boolean
```

```
    if asc(key) = 3 or asc(key) = 13 then
```

```
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
```

```
        bw.LittleEndian = true
```

```
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 8
```

```
        bw.WriteSingle(val(x_edit.text))
```

```
        bw.close
```

```
        scenario.vehicle_spawn(spawn_ind).x = val(x_edit.text)
```

```
        update_info
```

```
        new_box
```

```
        return true
```

```
    end
```

```
End Function
```

### **Vehi\_Spawn\_Control Control y\_updown:**

```
Sub Up()
```

```
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
```

```
    bw.LittleEndian = true
```

```
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 12
```

```
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).y + 1.0)
```

```
    bw.close
```

```
    scenario.vehicle_spawn(spawn_ind).y = scenario.vehicle_spawn(spawn_ind).y + 1.0
```

```
    update_info
```

```
    new_box
```

```
End Sub
```

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 12
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).y - 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).y = scenario.vehicle_spawn(spawn_ind).y - 1.0
    update_info
    new_box
End Sub

```

### **Vehi\_Spawn\_Control Control y\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 12
        bw.WriteSingle(val(y_edit.text))
        bw.close
        scenario.vehicle_spawn(spawn_ind).y = val(y_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **Vehi\_Spawn\_Control Control z\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 16
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).z - 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).z = scenario.vehicle_spawn(spawn_ind).z - 1.0
    update_info
    new_box
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 16
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).z + 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).z = scenario.vehicle_spawn(spawn_ind).z + 1.0
    update_info
    new_box
End Sub

```

### **Vehi\_Spawn\_Control Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)

```

```

        bw.LittleEndian = true
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 16
        bw.WriteSingle(val(z_edit.text))
        bw.close
        scenario.vehicle_spawn(spawn_ind).z = val(z_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **Vehi\_Spawn\_Control Control roll\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).roll - 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).roll = scenario.vehicle_spawn(spawn_ind).roll - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).roll + 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).roll = scenario.vehicle_spawn(spawn_ind).roll + 1.0
    update_info
End Sub

```

### **Vehi\_Spawn\_Control Control roll\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 28
        bw.WriteSingle(val(roll_edit.text))
        bw.close
        scenario.vehicle_spawn(spawn_ind).roll = val(roll_edit.text)
        update_info
        return true
    end
End Function

```

### **Vehi\_Spawn\_Control Control pitch\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 24
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).pitch - 1.0)
    bw.close

```

```

    scenario.vehicle_spawn(spawn_ind).pitch = scenario.vehicle_spawn(spawn_ind).pitch - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 24
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).pitch + 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).pitch = scenario.vehicle_spawn(spawn_ind).pitch + 1.0
    update_info
End Sub

```

### **Vehi\_Spawn\_Control Control pitch\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 24
        bw.WriteSingle(val(pitch_edit.text))
        bw.close
        scenario.vehicle_spawn(spawn_ind).pitch = val(pitch_edit.text)
        update_info
        return true
    end
End Function

```

### **Vehi\_Spawn\_Control Control yaw\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 20
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).yaw - 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).yaw = scenario.vehicle_spawn(spawn_ind).yaw - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.vehicle_spawn(spawn_ind).position + 20
    bw.WriteSingle(scenario.vehicle_spawn(spawn_ind).yaw + 1.0)
    bw.close
    scenario.vehicle_spawn(spawn_ind).yaw = scenario.vehicle_spawn(spawn_ind).yaw + 1.0
    update_info
End Sub

```

### **Vehi\_Spawn\_Control Control yaw\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)

```

```

        bw.LittleEndian = true
        bw.Position = scenario.vehicle_spawn(spawn_ind).position + 20
        bw.WriteSingle(val(yaw_edit.text))
        bw.close
        scenario.vehicle_spawn(spawn_ind).yaw = val(yaw_edit.text)
        update_info
        return true
    end
End Function
End Class

```

## **Class Weapon Spawn Control**

Inherits ContainerControl

### **Weapon\_Spawn\_Control.init:**

```

Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenario = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    for i as integer = 0 to UBound(Scenario.weapon_spawn)
        spawn_select.AddRow(scenario.weapon_spawn(i).name)
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub

```

### **Weapon\_Spawn\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.weapon_spawn(spawn_ind).x, _
    scenario.weapon_spawn(spawn_ind).y, scenario.weapon_spawn(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box
End Sub

```

### **Weapon\_Spawn\_Control.update\_info:**



```

Sub update_info()
    //break
    dim name_index as integer = Scenario.weapon_spawn(spawn_ind).name_index
    if name_index <> -1 then
        dim name_string as string = scenario.object_names(name_index).name
        name_static.text = name_string
    else
        name_static.Text = "Script name not defined"
    end

    rounds_left_edit.text = str(scenario.weapon_spawn(spawn_ind).rounds_left)
    rounds_loaded_edit.text = str(scenario.weapon_spawn(spawn_ind).rounds_loaded)

    x_edit.text = str(scenario.weapon_spawn(spawn_ind).x)
    y_edit.text = str(scenario.weapon_spawn(spawn_ind).y)
    z_edit.text = str(scenario.weapon_spawn(spawn_ind).z)

    roll_edit.Text = str(scenario.weapon_spawn(spawn_ind).roll)
    pitch_edit.Text = str(scenario.weapon_spawn(spawn_ind).pitch)
    yaw_edit.Text = str(scenario.weapon_spawn(spawn_ind).yaw)

    if scenario.weapon_spawn(spawn_ind).not_placed_auto then
        auto_placed_text.Text = "Auto Placed: No"
    else
        auto_placed_text.Text = "Auto Placed: Yes"
    end
    if scenario.weapon_spawn(spawn_ind).not_placed_easy then
        easy_placed_text.Text = "Placed on Easy: No"
    else
        easy_placed_text.Text = "Placed on Easy: Yes"
    end
    if scenario.weapon_spawn(spawn_ind).not_placed_normal then
        norm_placed_text.Text = "Placed on Normal: No"
    else
        norm_placed_text.Text = "Placed on Normal: Yes"
    end
    if scenario.weapon_spawn(spawn_ind).not_placed_hard then
        hard_placed_text.Text = "Placed on Hard: No"
    else
        hard_placed_text.Text = "Placed on Hard: Yes"
    end
End Sub
box As trimesh

f As folderitem

magic As Integer

scenario As scenario_file

spawn_ind As Integer

target As bsp_display

```

**Weapon\_Spawn\_Control Control TabPanel1:**

```
Sub Change()  
    update_info  
End Sub
```

**Weapon\_Spawn\_Control Control spawn\_select:**

```
Sub Change()  
    spawn_ind = me.ListIndex  
    new_box  
    update_info  
End Sub
```

**Weapon\_Spawn\_Control Control x\_updown:**

```
Sub Up()  
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = scenario.weapon_spawn(spawn_ind).position + 8  
    bw.WriteSingle(scenario.weapon_spawn(spawn_ind).x + 1.0)  
    bw.close  
    scenario.weapon_spawn(spawn_ind).x = scenario.weapon_spawn(spawn_ind).x + 1.0  
    update_info  
    new_box  
End Sub
```

```
Sub Down()  
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)  
    bw.LittleEndian = true  
    bw.Position = scenario.weapon_spawn(spawn_ind).position + 8  
    bw.WriteSingle(scenario.weapon_spawn(spawn_ind).x - 1.0)  
    bw.close  
    scenario.weapon_spawn(spawn_ind).x = scenario.weapon_spawn(spawn_ind).x - 1.0  
    update_info  
    new_box  
End Sub
```

**Weapon\_Spawn\_Control Control x\_edit:**

```
Function KeyDown(Key As String) As Boolean  
    if asc(key) = 3 or asc(key) = 13 then  
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)  
        bw.LittleEndian = true  
        bw.Position = scenario.weapon_spawn(spawn_ind).position + 8  
        bw.WriteSingle(val(x_edit.text))  
        bw.close  
        scenario.weapon_spawn(spawn_ind).x = val(x_edit.text)  
        update_info  
        new_box  
        return true  
    end  
End Function
```

**Weapon\_Spawn\_Control Control y\_updown:**

```
Sub Up()
```

```

dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).y + 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).y = scenario.weapon_spawn(spawn_ind).y + 1.0
update_info
new_box
End Sub

```

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 12
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).y - 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).y = scenario.weapon_spawn(spawn_ind).y - 1.0
update_info
new_box
End Sub

```

### **Weapon\_Spawn\_Control Control y\_edit:**

```

Function KeyDown(Key As String) As Boolean
if asc(key) = 3 or asc(key) = 13 then
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 12
bw.WriteSingle(val(y_edit.text))
bw.close
scenario.weapon_spawn(spawn_ind).y = val(y_edit.text)
update_info
new_box
return true
end
End Function

```

### **Weapon\_Spawn\_Control Control z\_updown:**

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 16
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).z - 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).z = scenario.weapon_spawn(spawn_ind).z - 1.0
update_info
new_box
End Sub

```

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 16
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).z + 1.0)
bw.close

```

```

scenario.weapon_spawn(spawn_ind).z = scenario.weapon_spawn(spawn_ind).z + 1.0
update_info
new_box
End Sub

```

### **Weapon\_Spawn\_Control Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.weapon_spawn(spawn_ind).position + 16
        bw.WriteSingle(val(z_edit.text))
        bw.close
        scenario.weapon_spawn(spawn_ind).z = val(z_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **Weapon\_Spawn\_Control Control roll\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.weapon_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.weapon_spawn(spawn_ind).roll - 1.0)
    bw.close
    scenario.weapon_spawn(spawn_ind).roll = scenario.weapon_spawn(spawn_ind).roll - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.weapon_spawn(spawn_ind).position + 28
    bw.WriteSingle(scenario.weapon_spawn(spawn_ind).roll + 1.0)
    bw.close
    scenario.weapon_spawn(spawn_ind).roll = scenario.weapon_spawn(spawn_ind).roll + 1.0
    update_info
End Sub

```

### **Weapon\_Spawn\_Control Control roll\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.weapon_spawn(spawn_ind).position + 28
        bw.WriteSingle(val(roll_edit.text))
        bw.close
        scenario.weapon_spawn(spawn_ind).roll = val(roll_edit.text)
        update_info
        return true
    end
end

```

End Function

### **Weapon\_Spawn\_Control Control pitch\_updown:**

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 24
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).pitch - 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).pitch = scenario.weapon_spawn(spawn_ind).pitch - 1.0
update_info
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 24
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).pitch + 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).pitch = scenario.weapon_spawn(spawn_ind).pitch + 1.0
update_info
```

End Sub

### **Weapon\_Spawn\_Control Control pitch\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.weapon_spawn(spawn_ind).position + 24
    bw.WriteSingle(val(pitch_edit.text))
    bw.close
    scenario.weapon_spawn(spawn_ind).pitch = val(pitch_edit.text)
    update_info
    return true
end
```

End Function

End Function

### **Weapon\_Spawn\_Control Control yaw\_updown:**

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 20
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).yaw - 1.0)
bw.close
scenario.weapon_spawn(spawn_ind).yaw = scenario.weapon_spawn(spawn_ind).yaw - 1.0
update_info
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.weapon_spawn(spawn_ind).position + 20
bw.WriteSingle(scenario.weapon_spawn(spawn_ind).yaw + 1.0)
```

```

    bw.close
    scenario.weapon_spawn(spawn_ind).yaw = scenario.weapon_spawn(spawn_ind).yaw + 1.0
    update_info
End Sub

```

### **Weapon\_Spawn\_Control yaw\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.weapon_spawn(spawn_ind).position + 20
        bw.WriteSingle(val(yaw_edit.text))
        bw.close
        scenario.weapon_spawn(spawn_ind).yaw = val(yaw_edit.text)
        update_info
        return true
    end
End Function
End Class

```

## **Class NetEquip Spawn Control**

Inherits ContainerControl

### **NetEquip\_Spawn\_Control.init:**

```

Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenario = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    //consider listing only the types of the spawn points and then having another list of each one
    //too damn many
    for i as integer = 0 to UBound(Scenario.MP_equip)
        spawn_select.AddRow(scenario.MP_equip(i).dep_name)
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub

```

### **NetEquip\_Spawn\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.MP_equip(spawn_ind).x, _
    scenario.MP_equip(spawn_ind).y, scenario.MP_equip(spawn_ind).z, 1, false)

```

```

box.HasVertexColors = false
box.NullShader = false
box.RenderBackFaces = true
box.Material.HasDiffuseColor = true
box.Material.DiffuseColor = Target.object_color

target.object_box = box
target.new_box
End Sub

```

### **NetEquip\_Spawn\_Control.update\_info:**

```

Sub update_info()
    team_index_edit.text = str(scenario.MP_equip(spawn_ind).team_index)
    time_static.text = "Spawn Time: " + str(scenario.MP_equip(spawn_ind).spawn_time)

    x_edit.text = str(scenario.MP_equip(spawn_ind).x)
    y_edit.text = str(scenario.MP_equip(spawn_ind).y)
    z_edit.text = str(scenario.MP_equip(spawn_ind).z)

    roll_edit.Text = str(scenario.MP_equip(spawn_ind).yaw)

    dim type_string as string
    select case Scenario.MP_equip(spawn_ind).type0
    case 0
        type_string = "None"
    case 1
        type_string = "CTF"
    case 2
        type_string = "Slayer"
    case 3
        type_string = "Oddball"
    case 4
        type_string = "King of the Hill"
    case 5
        type_string = "Race"
    case 6
        type_string = "Terminator"
    case 7
        type_string = "Stub"
    case 12
        type_string = "All Gametypes"
    case 13
        type_string = "All except CTF"
    case 14
        type_string = "All except Race & CTF"
    case else
        type_string = "Unknown"
    end select

    type_text1.text = "Type1: " + type_string

    select case Scenario.MP_equip(spawn_ind).type1
    case 0
        type_string = "None"

```

```

case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

```

```

type_text2.text = "Type2: " + type_string

```

```

select case Scenario.MP_equip(spawn_ind).type2
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

```

```

type_text3.text = "Type3: " + type_string

```



```

select case Scenario.MP_equip(spawn_ind).type3
case 0
    type_string = "None"
case 1
    type_string = "CTF"
case 2
    type_string = "Slayer"
case 3
    type_string = "Oddball"
case 4
    type_string = "King of the Hill"
case 5
    type_string = "Race"
case 6
    type_string = "Terminator"
case 7
    type_string = "Stub"
case 12
    type_string = "All Gametypes"
case 13
    type_string = "All except CTF"
case 14
    type_string = "All except Race & CTF"
case else
    type_string = "Unknown"
end select

type_text4.text = "Type4: " + type_string

class2_static.text = scenario.MP_equip(spawn_ind).itmctag
string_static.text = Scenario.MP_equip(spawn_ind).dep_name
End Sub
box As trimesh

f As folderitem

magic As Integer

scenario As scenario_file

spawn_ind As Integer

target As bsp_display

```

### **NetEquip\_Spawn\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **NetEquip\_Spawn\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex

```

```
new_box
update_info
End Sub
```

### **NetEquip\_Spawn\_Control Control x\_updown:**

```
Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40
    bw.WriteSingle(scenario.MP_equip(spawn_ind).x - 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).x = scenario.MP_equip(spawn_ind).x - 1.0
    update_info
    new_box
End Sub
```

```
Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40
    bw.WriteSingle(scenario.MP_equip(spawn_ind).x + 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).x = scenario.MP_equip(spawn_ind).x + 1.0
    update_info
    new_box
End Sub
```

### **NetEquip\_Spawn\_Control Control x\_edit:**

```
Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.MP_equip(spawn_ind).position + &h40
        bw.WriteSingle(val(x_edit.text))
        bw.close
        scenario.MP_equip(spawn_ind).x = val(x_edit.text)
        update_info
        new_box
        return true
    end
End Function
```

### **NetEquip\_Spawn\_Control Control y\_updown:**

```
Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 4
    bw.WriteSingle(scenario.MP_equip(spawn_ind).y + 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).y = scenario.MP_equip(spawn_ind).y + 1.0
    update_info
    new_box
End Sub
```

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 4
    bw.WriteSingle(scenario.MP_equip(spawn_ind).y - 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).y = scenario.MP_equip(spawn_ind).y - 1.0
    update_info
    new_box
End Sub

```

### **NetEquip\_Spawn\_Control Control y\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 4
        bw.WriteSingle(val(y_edit.text))
        bw.close
        scenario.MP_equip(spawn_ind).y = val(y_edit.text)
        update_info
        new_box
        return true
    end
End Function

```

### **NetEquip\_Spawn\_Control Control z\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 8
    bw.WriteSingle(scenario.MP_equip(spawn_ind).z - 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).z = scenario.MP_equip(spawn_ind).z - 1.0
    update_info
    new_box
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 8
    bw.WriteSingle(scenario.MP_equip(spawn_ind).z + 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).z = scenario.MP_equip(spawn_ind).z + 1.0
    update_info
    new_box
End Sub

```

### **NetEquip\_Spawn\_Control Control z\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then

```

```

        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 8
        bw.WriteSingle(val(z_edit.text))
        bw.close
        scenario.MP_equip(spawn_ind).z = val(z_edit.text)
        update_info
        new_box
        return true
    end
End Function
NetEquip_Spawn_Control Control roll_updown:

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 12
    bw.WriteSingle(scenario.MP_equip(spawn_ind).yaw - 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).yaw = scenario.MP_equip(spawn_ind).yaw - 1.0
    update_info
End Sub

```

```

Sub Up()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 12
    bw.WriteSingle(scenario.MP_equip(spawn_ind).yaw + 1.0)
    bw.close
    scenario.MP_equip(spawn_ind).yaw = scenario.MP_equip(spawn_ind).yaw + 1.0
    update_info
End Sub

```

### **NetEquip\_Spawn\_Control Control roll\_edit:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        dim bw as BinaryStream = f.OpenAsBinaryFile(true)
        bw.LittleEndian = true
        bw.Position = scenario.MP_equip(spawn_ind).position + &h40 + 12
        bw.WriteSingle(val(roll_edit.text))
        bw.close
        scenario.MP_equip(spawn_ind).yaw = val(roll_edit.text)
        update_info
        return true
    end
End Function
End Class

```

## **Class NetFlag\_Spawn\_Control**

Inherits ContainerControl

### **NetFlag\_Spawn\_Control.init:**

```

Sub init(byref in_scen as scenario_file, in_f as folderItem, in_magic as integer, byref in_target as bsp_display)
    scenarior = in_scen
    f = in_f
    magic = in_magic
    target = in_target

    spawn_select.DeleteAllRows

    //ok, now list all of the existing spawn points
    //consider listing only the types of the spawn points and then having another list of each one
    //too damn many
    for i as integer = 0 to UBound(Scenario.MP_flag)
        spawn_select.AddRow(scenario.MP_flag(i).name)
    next
    spawn_select.ListIndex = 0
    //and now initialize the first selection
    spawn_ind = 0
    new_box
    update_info
End Sub

```

### **NetFlag\_Spawn\_Control.new\_box:**

```

Sub new_box()
    //this will eventually get changed to get the actual model of the object

    box = Filemanip_old.return_box(scenario.MP_flag(spawn_ind).x, _
    scenario.MP_flag(spawn_ind).y, scenario.MP_flag(spawn_ind).z, 1, false)

    box.HasVertexColors = false
    box.NullShader = false
    box.RenderBackFaces = true
    box.Material.HasDiffuseColor = true
    box.Material.DiffuseColor = Target.object_color

    target.object_box = box
    target.new_box
End Sub

```

### **NetFlag\_Spawn\_Control.update\_info:**

```

Sub update_info()
    team_index_edit.text = str(scenario.MP_flag(spawn_ind).teamindex)

    x_edit.text = str(scenario.MP_flag(spawn_ind).x)
    y_edit.text = str(scenario.MP_flag(spawn_ind).y)
    z_edit.text = str(scenario.MP_flag(spawn_ind).z)

    roll_edit.Text = str(scenario.MP_flag(spawn_ind).yaw)

    dim type_string as string = "Type: unknown"
    select case Scenario.MP_flag(spawn_ind).type
    case 0
        type_string = "CTF - flag"
    case 1
        type_string = "CTF - vehicle"
    end select

```

```

case 2
    type_string = "Oddball"
case 3
    type_string = "Race – trak"
case 4
    type_string = "Race – vehicle"
case 5
    type_string = "Vegas – bank"
case 6
    type_string = "Teleporter – entrance"
case 7
    type_string = "Teleporter – exit"
case 8
    type_string = "Hill"
end select

type_static.text = type_string

class2_static.text = scenario.MP_flag(spawn_ind).dep.tag
string_static.text = Scenario.MP_flag(spawn_ind).dep_name
End Sub
box As trimesh

f As folderitem

magic As Integer

scenario As scenario_file

spawn_ind As Integer

target As bsp_display

```

### **NetFlag\_Spawn\_Control Control TabPanel1:**

```

Sub Change()
    update_info
End Sub

```

### **NetFlag\_Spawn\_Control Control spawn\_select:**

```

Sub Change()
    spawn_ind = me.ListIndex
    new_box
    update_info
End Sub

```

### **NetFlag\_Spawn\_Control Control x\_updown:**

```

Sub Down()
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_flag(spawn_ind).position
    bw.WriteSingle(scenario.MP_flag(spawn_ind).x – 1.0)
    bw.close

```

```

scenario.MP_flag(spawn_ind).x = scenario.MP_flag(spawn_ind).x - 1.0
update_info
new_box
End Sub

```

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position
bw.WriteSingle(scenario.MP_flag(spawn_ind).x + 1.0)
bw.close
scenario.MP_flag(spawn_ind).x = scenario.MP_flag(spawn_ind).x + 1.0
update_info
new_box
End Sub

```

### **NetFlag\_Spawn\_Control Control x\_edit:**

```

Function KeyDown(Key As String) As Boolean
if asc(key) = 3 or asc(key) = 13 then
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position
bw.WriteSingle(val(x_edit.text))
bw.close
scenario.MP_flag(spawn_ind).x = val(x_edit.text)
update_info
new_box
return true
end
End Function

```

### **NetFlag\_Spawn\_Control Control y\_updown:**

```

Sub Up()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 4
bw.WriteSingle(scenario.MP_flag(spawn_ind).y + 1.0)
bw.close
scenario.MP_flag(spawn_ind).y = scenario.MP_flag(spawn_ind).y + 1.0
update_info
new_box
End Sub

```

```

Sub Down()
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 4
bw.WriteSingle(scenario.MP_flag(spawn_ind).y - 1.0)
bw.close
scenario.MP_flag(spawn_ind).y = scenario.MP_flag(spawn_ind).y - 1.0
update_info
new_box
End Sub

```

NetFlag\_Spawn\_Control Control y\_edit:

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_flag(spawn_ind).position + 4
    bw.WriteSingle(val(y_edit.text))
    bw.close
    scenario.MP_flag(spawn_ind).y = val(y_edit.text)
    update_info
    new_box
    return true
end
```

End Function

**NetFlag\_Spawn\_Control Control z\_updown:**

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 8
bw.WriteSingle(scenario.MP_flag(spawn_ind).z - 1.0)
bw.close
scenario.MP_flag(spawn_ind).z = scenario.MP_flag(spawn_ind).z - 1.0
update_info
new_box
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 8
bw.WriteSingle(scenario.MP_flag(spawn_ind).z + 1.0)
bw.close
scenario.MP_flag(spawn_ind).z = scenario.MP_flag(spawn_ind).z + 1.0
update_info
new_box
```

End Sub

**NetFlag\_Spawn\_Control Control z\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_flag(spawn_ind).position + 8
    bw.WriteSingle(val(z_edit.text))
    bw.close
    scenario.MP_flag(spawn_ind).z = val(z_edit.text)
    update_info
    new_box
    return true
end
```

End Function



NetFlag\_Spawn\_Control Control roll\_updown:

Sub Down()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 12
bw.WriteSingle(scenario.MP_flag(spawn_ind).yaw - 1.0)
bw.close
scenario.MP_flag(spawn_ind).yaw = scenario.MP_flag(spawn_ind).yaw - 1.0
update_info
```

End Sub

Sub Up()

```
dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
bw.Position = scenario.MP_flag(spawn_ind).position + 12
bw.WriteSingle(scenario.MP_flag(spawn_ind).yaw + 1.0)
bw.close
scenario.MP_flag(spawn_ind).yaw = scenario.MP_flag(spawn_ind).yaw + 1.0
update_info
```

End Sub

**NetFlag\_Spawn\_Control Control roll\_edit:**

Function KeyDown(Key As String) As Boolean

```
if asc(key) = 3 or asc(key) = 13 then
    dim bw as BinaryStream = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = scenario.MP_flag(spawn_ind).position + 12
    bw.WriteSingle(val(roll_edit.text))
    bw.close
    scenario.MP_flag(spawn_ind).yaw = val(roll_edit.text)
    update_info
    return true
end
```

End Function

End Class

## **Class BSP\_class**

**BSP\_class.read:**

Sub read(byref br as binaryStream, offset as integer, magic as integer, in\_color as color = &c0AFF0A)

```
br.LittleEndian = true
br.Position = offset
```

```
local_offset = br.ReadUInt32
```

```
br.Position = offset + (local_offset - magic)
dim temp_dep as new Dependency
lightmaps = temp_dep.read(br)
```

```

for i as integer = 1 to &h25
    dim temp as integer = br.ReadUInt32
next

dim temp_ref as new Reflexive
shaders = temp_ref.read(br)
temp_ref = new Reflexive
collbspheader = temp_ref.read(br)
temp_ref = new Reflexive
nodes = temp_ref.read(br)

for i as integer = 1 to 6
    dim temp as integer = br.ReadUInt32
next

temp_ref = new Reflexive
leaves = temp_ref.read(br)
temp_ref = new Reflexive
leafsurfaces = temp_ref.read(br)
temp_ref = new Reflexive
submeshTriIndices = temp_ref.read(br)
temp_ref = new Reflexive
submeshHeader = temp_ref.read(br)

br.Position = submeshTriIndices.offset + offset - magic
redim subTri_Ind_1(-1)
redim subTri_Ind_2(-1)
redim subTri_Ind_3(-1)
for i as integer = 1 to submeshTriIndices.count
    subTri_Ind_1.Append br.ReadShort
    subTri_Ind_2.Append br.ReadShort
    subTri_Ind_3.Append br.ReadShort
next

//method that's lame and reads all of the submesh headers mwahahaha!
redim submesh_headers(-1)
dim SMH_offsets(-1) as integer
br.Position = submeshHeader.offset + offset - magic
for i as integer = 1 to submeshHeader.count
    dim lightmapIndex as short = br.ReadShort
    dim unk1 as short = br.ReadShort
    for j as integer = 1 to 4
        dim temp as integer = br.ReadUInt32
    next
    dim Material_1 as new Reflexive
    Material_1 = Material_1.read(br)
    for j as integer = 0 to Material_1.count - 1
        SMH_offsets.Append Material_1.offset - magic + offset + (256*j)
    next
next

//not really the way it ought to be
for i as integer = 0 to UBound(SMH_offsets)

```

```

    br.Position = SMH_offsets(i)
    dim temp_sub_head as new submesh_header
    temp_sub_head.read(br, magic, offset)
    submesh_headers.Append temp_sub_head
next

//render the visible meshes
redim bsp_tris(-1)
for i as integer = 0 to UBound(submesh_headers)
    return_tm = new trimesh
    return_tm.renderbackfaces = true
    return_tm.NullShader = false
    return_tm.Material.HasDiffuseColor = true
    return_tm.Material.DiffuseColor = in_color
    return_tm.VertexCount = submesh_headers(i).VertexCount1
    for j as integer = 0 to submesh_headers(i).VertexCount1-1
        return_tm.VertexPositions.SetXYZ(j, submesh_headers(i).x(j), submesh_headers(i).y(j),
            submesh_headers(i).z(j))
    next
    //might comment this out
    return_tm.HasVertexNormals = true
    for j as integer = 0 to submesh_headers(i).VertexCount1-1
        return_tm.VertexNormals.SetXYZ(j, submesh_headers(i).normals_x(j), _
            submesh_headers(i).normals_y(j), submesh_headers(i).normals_z(j))
    next
    //end of might
    return_tm.TriangleCount = submesh_headers(i).vertIndexCount
    for j as integer = 0 to submesh_headers(i).vertIndexCount-1
        return_tm.Triangles.SetABC(j, _
            subTri_Ind_1(submesh_headers(i).vertIndexOffset + j), _
            subTri_Ind_2(submesh_headers(i).vertIndexOffset + j), _
            subTri_Ind_3(submesh_headers(i).vertIndexOffset + j) )
    next
    bsp_tris.Append return_tm
next
End Sub
bsp_tris(-1) As trimesh

collbspheader As reflexive

leafsurfaces As reflexive

leaves As reflexive

lightmaps As dependency

local_offset As Integer

nodes As reflexive

return_tm As trimesh

```

shaders As reflexive

submeshHeader As reflexive

submeshTriIndices As reflexive

submesh\_headers(-1) As submesh\_header

subTri\_Ind\_1(-1) As short

subTri\_Ind\_2(-1) As short

subTri\_Ind\_3(-1) As short

End Class

## **Module BSP\_stuff**

### **BSP\_stuff.read\_header:**

Sub read\_header(byref br as binaryStream, offset as integer, magic as integer)

br.LittleEndian = true

br.Position = offset

dim local\_offset as integer = br.ReadUInt32

br.Position = offset + (local\_offset - magic)

dim lightmaps as new Dependency

lightmaps = lightmaps.read(br)

for i as integer = 1 to &h25

dim temp as integer = br.ReadUInt32

next

dim shaders as new Reflexive

shaders = shaders.read(br)

dim collbspheader as new Reflexive

collbspheader = collbspheader.read(br)

dim nodes as new Reflexive

nodes = nodes.read(br)

for i as integer = 1 to 6

dim temp as integer = br.ReadUInt32

next

dim leaves as new Reflexive

leaves = leaves.read(br)

dim leafsurfaces as new Reflexive

leafsurfaces = leafsurfaces.read(br)

dim submeshTriIndices as new Reflexive

submeshTriIndices = submeshTriIndices.read(br)

dim submeshHeader as new Reflexive

submeshHeader = submeshHeader.read(br)

```

br.Position = submeshTriIndices.offset + offset - magic
dim subTri_Ind_1(-1) as short
dim subTri_Ind_2(-1) as short
dim subTri_Ind_3(-1) as short
for i as integer = 1 to submeshTriIndices.count
    subTri_Ind_1.Append br.ReadShort
    subTri_Ind_2.Append br.ReadShort
    subTri_Ind_3.Append br.ReadShort
next

'br.Position = submeshHeader.offset + offset - magic
'dim lightmapIndex as short = br.ReadShort
'dim unk1 as short = br.ReadShort
'for i as integer = 1 to 4
'dim temp as integer = br.ReadUInt32
'next
'dim Material_1 as new Reflexive
'Material_1 = Material_1.read(br)
,

'br.Position = Material_1.offset + offset - magic
,

'dim submesh_headers(-1) as submesh_header
'for i as integer = 1 to Material_1.count
'dim temp_sub_header as new submesh_header
'temp_sub_header.read(br, magic, offset)
'submesh_headers.Append temp_sub_header
'next

//method that's lame and reads all of the submesh headers mwahahaha!
dim submesh_headers(-1) as submesh_header
dim SMH_offsets(-1) as integer
br.Position = submeshHeader.offset + offset - magic
for i as integer = 1 to submeshHeader.count
    dim lightmapIndex as short = br.ReadShort
    dim unk1 as short = br.ReadShort
    for j as integer = 1 to 4
        dim temp as integer = br.ReadUInt32
    next
    dim Material_1 as new Reflexive
    Material_1 = Material_1.read(br)
    for j as integer = 0 to Material_1.count - 1
        SMH_offsets.Append Material_1.offset - magic + offset + (256*j)
    next
next

//not really the way it ought to be
for i as integer = 0 to UBound(SMH_offsets)
    br.Position = SMH_offsets(i)
    dim temp_sub_head as new submesh_header
    temp_sub_head.read(br, magic, offset)
    submesh_headers.Append temp_sub_head
next

```

```

//break

//get a lot of points to represent the map
dim bsp_tris(-1) as trimesh
for i as integer = 0 to UBound(submesh_headers)
    return_tm = new trimesh
    return_tm.renderbackfaces = true
    return_tm.NullShader = true
    return_tm.VertexCount = submesh_headers(i).VertexCount1
    for j as integer = 0 to submesh_headers(i).VertexCount1-1
        return_tm.VertexPositions.SetXYZ(j, submesh_headers(i).x(j), submesh_headers(i).y(j),
            submesh_headers(i).z(j))
    next
    return_tm.TriangleCount = submesh_headers(i).vertIndexCount
    for j as integer = 0 to submesh_headers(i).vertIndexCount-1
        return_tm.Triangles.SetABC(j, _
            subTri_Ind_1(submesh_headers(i).vertIndexOffset + j), _
            subTri_Ind_2(submesh_headers(i).vertIndexOffset + j), _
            subTri_Ind_3(submesh_headers(i).vertIndexOffset + j) )
    next
    bsp_tris.Append return_tm
next

'//display the stuff
'dim w as new TEST_3d
'w.get_mod2_submodel(bsp_tris)
'w.x_slider.Value = 120
'w.y_slider.Value = -178
'w.show

//break
End Sub
return_tm As trimesh

```

## **BSP\_stuff Note: getting the scenario bsp data**

getting the scenario bsp data

```

dim a as new Scenario_Header
a = scenario_header_read(map)
dim br as BinaryStream = map.f.OpenAsBinaryFile
br.LittleEndian = true
br.position = a.StructBsp.offset - map.primarymagic

```

```

dim BspStart() as UInt32
dim BspSize() as UInt32
dim BspMagic() as UInt32
dim Zero1() as UInt32
dim bsptag() as string
dim namePtr() as uint32
dim unknown2() as uint32
dim tagid() as int32

```

```

for i as integer = 1 to a.StructBsp.count
  BspStart.append br.readuint32
  BspSize.append br.ReadUInt32
  BspMagic.Append br.ReadUInt32
  Zero1.Append br.ReadUInt32
  bsptag.Append br.read(4)
  namePtr.append br.ReadUInt32
  unknown2.Append br.ReadUInt32
  tagid.Append br.ReadInt32
next

break
End Module

```

## **Class submesh\_header**

### **submesh\_header.read:**

Sub read(byref br as binaryStream, magic as integer, offset as integer)

```

dim temp_dep as new Dependency
shader_tag = temp_dep.read(br)
unkzero2 = br.ReadUInt32
vertIndexOffset = br.ReadUInt32
vertIndexCount = br.ReadUInt32
redim centroid(-1)
for i as integer = 1 to 3
  centroid.Append br.ReadSingle
next
redim ambientColor(-1)
for i as integer = 1 to 3
  ambientColor.Append br.ReadSingle
next
ambientColor_color = rgb(ambientColor(0) * 255, ambientColor(1) * 255, ambientColor(2) * 255)
distLightCount = br.ReadUInt32
redim distLight1(-1)
for i as integer = 1 to 6
  distLight1.Append br.ReadSingle
next
redim distLight2(-1)
for i as integer = 1 to 6
  distLight2.Append br.ReadSingle
next
redim unkFloat2(-1)
for i as integer = 1 to 3
  unkFloat2.Append br.ReadSingle
next
redim reflective_tint(-1)
for i as integer = 1 to 4
  reflective_tint.Append br.ReadSingle
next
redim ShadowVector(-1)

```

```

for i as integer = 1 to 3
    ShadowVector.Append br.ReadSingle
next
redim ShadowColor(-1)
for i as integer = 1 to 3
    ShadowColor.append br.ReadSingle
next
ShadowColor_color = rgb(shadowColor(0) * 255, shadowColor(1) * 255, shadowColor(2) * 255)
redim Plane(-1)
for i as integer = 1 to 4
    Plane.append br.ReadSingle
next
unkFlag2 = br.ReadUInt32
unkCount = br.ReadUInt32
VertexCount1 = br.ReadUInt32
unkZero4 = br.ReadUInt32
VertexOffset = br.ReadUInt32
Vert_Reflexive = br.ReadUInt32
unkAlways3 = br.ReadUInt32
VertexCount2 = br.ReadUInt32
unkZero9 = br.ReadUInt32
unkLightMapOffset = br.ReadUInt32
CompVert_Reflexive = br.ReadUInt32
redim unkZero5(-1)
for i as integer = 1 to 2
    unkZero5.Append br.ReadUInt32
next
someOffset1 = br.ReadUInt32
PcVertexDataOffset = br.ReadUInt32 - magic + offset
unkZero6 = br.ReadUInt32
CompVertBufferSize = br.ReadUInt32
unkZero7 = br.ReadUInt32
someOffset2 = br.ReadUInt32
VertexDataOffset = br.ReadUInt32
unkZero8 = br.ReadUInt32

//now it's time to read all of the verts associated with it
br.Position = PcVertexDataOffset
redim x(-1)
redim y(-1)
redim z(-1)
redim binormals(-1)
redim normals_x(-1)
redim normals_y(-1)
redim normals_z(-1)
redim tangents(-1)
redim uv(-1)
for i as integer = 1 to VertexCount1
    x.append br.ReadSingle
    y.append br.ReadSingle
    z.append br.ReadSingle

    //uncompressed verts
    normals_x.Append br.ReadSingle

```



```

    normals_y.Append br.ReadSingle
    normals_z.Append br.ReadSingle

    for k as integer = 1 to 3
        binormals.Append br.ReadSingle
    next
    for k as integer = 1 to 3
        tangents.Append br.ReadSingle
    next
    dim uv(-1) as single
    for k as integer = 1 to 2
        uv.Append br.ReadSingle
    next
    'bsp_tris.Append return_box(x(ubound(x)),y(ubound(y)),z(ubound(z)))
next
End Sub
ambientColor(-1) As single

ambientColor_color As color

binormals(-1) As Integer

centroid(-1) As single

CompVertexBufferSize As uint32

CompVert_Reflexive As uint32

distLight1(-1) As single

distLight2(-1) As single

distLightCount As UInt32

normals_x(-1) As single

normals_y(-1) As single

normals_z(-1) As single

PcVertexDataOffset As uint32

Plane(-1) As single

reflective_tint(-1) As single

shader_tag As dependency

shadowColor(-1) As single

shadowColor_color As color

shadowVector(-1) As single

```

someOffset1 As uint32

someOffset2 As uint32

tangents(-1) As single

**submesh\_header.unkAlways3:**

unkAlways3 As uint32

//well not on a mac map from what I've seen

unkCount As uint32

unkFlag2 As uint32

unkFloat2(-1) As single

unkLightMapOffset As uint32

unkzero2 As uint32

unkZero4 As uint32

unkZero5(-1) As uint32

unkZero6 As uint32

unkZero7 As uint32

unkZero8 As uint32

unkZero9 As uint32

uv(-1) As single

VertexCount1 As uint32

vertexCount2 As uint32

VertexDataOffset As uint32

VertexOffset As uint32

vertIndexCount As uint32

vertIndexOffset As uint32

Vert\_reflexive As uint32

x(-1) As single

y(-1) As single

z(-1) As single

data size

256 bytes

End Class

## Class Scenario\_file

### **Scenario\_file.get\_dependency\_name:**

Function get\_dependency\_name(ident as integer) As string

dim return\_str as string

if map.tags\_by\_ident.HasKey(ident) then

return\_str = map.tags\_list(map.tags\_by\_ident.Value(ident)).fullname

else

return\_str = "nulled out"

end

return return\_str

End Function

### **Scenario\_file.init:**

Sub init(byref in\_map as mapfile)

map = in\_map

header = scenario\_header\_read(map, map.base\_tag)

dim br as BinaryStream = map.f.OpenAsBinaryFile

br.LittleEndian = true

//get some object names

br.Position = header.ObjectNames.offset - map.primarymagic

for i as integer = 1 to header.ObjectNames.count

dim temp as new STRUCT\_OBJECT\_NAMES

temp.readStruct(br)

object\_names.Append(temp)

next

//let's get some playerspawns now

br.Position = header.PlayerSpawn.offset - map.primarymagic

for i as integer = 1 to header.PlayerSpawn.count

dim temp as new STRUCT\_PLAYER\_SPAWN

temp.readStruct(br)

player\_spawn.Append temp

next

//let's get some MP\_flags now

br.Position = header.MultiplayerFlags.offset - map.primarymagic

redim MP\_flag(-1)

for i as integer = 1 to Header.MultiplayerFlags.count

dim temp as new STRUCT\_MP\_FLAG

temp.readStruct(br)

temp.dep\_name = get\_dependency\_name(temp.dep.TagID)

```

    MP_flag.Append(temp)
next

//let's get MP_equip locations now
br.Position = header.MpEquip.offset - map.primarymagic
redim MP_equip(-1)
for i as integer = 1 to header.MpEquip.count
    dim temp as new STRUCT_MP_EQUIP
    temp.readstruct(br)
    temp.dep_name = get_dependency_name(temp.itmc.TagID)
    MP_equip.Append temp
next

//let's do vehicles now
//first get all of the references
br.Position = header.VehicleRef.offset - map.primarymagic
redim vehicle_ref(-1)
for i as integer = 1 to header.VehicleRef.count
    dim temp as new STRUCT_VEHICLE_REF
    temp.readStruct(br)
    vehicle_ref.Append temp
next
//then get all of the vehicle spawns
br.Position = header.Vehicle.offset - map.primarymagic
redim vehicle_spawn(-1)
for i as integer = 1 to header.Vehicle.count
    dim temp as new STRUCT_VEHICLE_SPAWN
    temp.readStruct(br)
    if temp.type <> -1 then
        temp.name = get_dependency_name(vehicle_ref(temp.type).vehicle.tagid)
    else
        temp.name = "Unknown"
    end
    vehicle_spawn.Append temp
next

//let's do scenery now
//first get all of the references
br.Position = header.SceneryRef.offset - map.primarymagic
redim Scenery_ref(-1)
for i as integer = 1 to header.SceneryRef.count
    dim temp as new STRUCT_PALLETTE
    temp.readStruct(br)
    Scenery_ref.Append temp
next
//then get all of the spawns
br.Position = header.scenery.offset - map.primarymagic
redim scenery_spawn(-1)
for i as integer = 1 to header.scenery.count
    dim temp as new STRUCT_SCENERY_SPAWN
    temp.readStruct(br)
    if temp.type_index <> -1 then
        temp.name = get_dependency_name(scenery_ref(temp.type_index).tagid)
    else

```

```

        temp.name = "Unknown"
    end
    if temp.name_index <> -1 then
        temp.name_script = object_names(temp.name_index).name
    else
        temp.name_script = "Script name not defined"
    end
    scenery_spawn.Append temp
next

//let's do weapons now
//first get all of the references
br.Position = header.WeapRef.offset - map.primarymagic
redim Weapon_ref(-1)
for i as integer = 1 to header.WeapRef.count
    dim temp as new STRUCT_PALLETTE
    temp.readStruct(br)
    Weapon_ref.Append temp
next
//then get all of the spawns
br.Position = header.Weap.offset - map.primarymagic
redim weapon_spawn(-1)
for i as integer = 1 to header.weap.count
    dim temp as new STRUCT_WEAPON_SPAWN
    temp.readStruct(br)
    if temp.type_index <> -1 then
        temp.name = get_dependency_name(weapon_ref(temp.type_index).tagid)
    else
        temp.name = "Unknown"
    end
    if temp.name_index <> -1 then
        temp.name_script = object_names(temp.name_index).name
    else
        temp.name_script = "Script name not defined"
    end
    weapon_spawn.Append temp
next

//let's do bipeds now
//first get all of the references
br.Position = header.BipedRef.offset - map.primarymagic
redim Biped_ref(-1)
for i as integer = 1 to header.BipedRef.count
    dim temp as new STRUCT_PALLETTE
    temp.readStruct(br)
    biped_ref.Append temp
next
//then get all of the spawns
br.Position = header.biped.offset - map.primarymagic
redim biped(-1)
for i as integer = 1 to header.biped.count
    dim temp as new STRUCT_BIPED
    temp.readStruct(br)
    if temp.type_index <> -1 then

```

```

        temp.name = get_dependency_name(biped_ref(temp.type_index).tagid)
    else
        temp.name = "Unknown"
    end
    if temp.name_index <> -1 then
        temp.name_script = object_names(temp.name_index).name
    else
        temp.name_script = "Script name not defined"
    end
    biped.Append temp
next

//break

//am going to have to figure out some methods for going into the vehi, itmc and matg (and others) dependencies
//once in there, determine the primary mod2 Dependency
//then go to that mod2 and get a LOD off of it for use in the viewer
//this is all further down the road

//break
End Sub
Actor_variant_ref() As struct_acTOR_VARIANT_REF

AI_animation_ref() As struct_AI_ANIMATION_REF

AI_recording_ref() As struct_AI_RECORDING_REF

AI_Script_ref As struct_AI_SCRIPT_REF

biped() As struct_BIPED

biped_ref() As struct_PALLETTE

BSP_trigger() As struct_BSP_TRIGGER

control() As struct_CONTROL

decal() As struct_DECAL

decal_ref() As struct_DECAL_REF

device_group() As struct_DEVICE_GROUP

encounter() As struct_ENCOUNTER

encounter_info() As struct_ENCOUNTER_INFO

enc_squad() As struct_ENC_SQUAD

enc_squad_spawn() As struct_ENC_SQUAD_SPAWN

equip() As struct_EQUIP

globals() As struct_GLOBALS

```

header As scenario\_Header

light\_fixture() As struct\_LIGHT\_FIXTURE

machine() As struct\_MACHINE

map As mapfile

move\_positions() As struct\_MOVE\_POSITIONS

MP\_equip() As Struct\_MP\_EQUIP

MP\_flag() As Struct\_MP\_FLAG

object\_names() As Struct\_OBJECT\_NAMES

pallette() As struct\_PALLETTE

player\_spawn() As Struct\_PLAYER\_SPAWN

player\_starting\_profile() As struct\_PLAYER\_STARTING\_PROFILE

Profile\_placement() As struct\_PROFILE\_PLACEMENT

references() As struct\_REFERENCES

scenery\_ref() As Struct\_PALLETTE

scenery\_spawn() As Struct\_SCENERY\_SPAWN

Script\_Trigger() As struct\_SCRIPT\_TRIGGER

shader\_index() As struct\_SHADER\_INDEX

skybox() As struct\_SKYBOX

sound\_scenery() As struct\_SOUND\_SCENERY

starting\_equip() As struct\_STARTING\_EQUIP

Trigger\_Volumes() As struct\_TRIGGER\_VOLUMES

vehicle\_ref() As STRUCT\_VEHICLE\_REF

vehicle\_spawn() As struct\_VEHICLE\_SPAWN

weapon\_ref() As STRUCT\_PALLETTE

weapon\_spawn() As struct\_WEAPON\_SPAWN

End Class

## **Class Scenario\_Header**

ActorVariantRef As Reflexive

AiAnimationRefs As Reflexive

AiConversations As Reflexive

AiRecordingRefs As Reflexive

Animations As Reflexive

Biped As Reflexive

BipedRef As Reflexive

BspSwitchTrigger As Reflexive

ChildScenarios As Reflexive

CommandLists As Reflexive

Commands As Reflexive

Control As Reflexive

ControlRef As Reflexive

CutsceneCameraPoi As Reflexive

CutsceneFlags As Reflexive

CutsceneTitles As Reflexive

Decals As Reflexive

DecalsRef As Reflexive

DetailObjCollRef As Reflexive

DeviceGroups As Reflexive

EditorScenarioSize As Integer

Encounters As Reflexive

Equip As Reflexive

EquipRef As Reflexive

GlobalsVerified As Reflexive

LightFixture As Reflexive

LightFixtureRef As Reflexive



Lines As Reflexive

Machine As Reflexive

MachineRef As Reflexive

MpEquip As Reflexive

MultiplayerFlags As Reflexive

ObjectNames As Reflexive

Participants As Reflexive

Platoons As Reflexive

PlayerSpawn As Reflexive

PlayerStartingProfile As Reflexive

pointertoendofindex As Integer

pointertoindex As Integer

Points As Reflexive

Scenery As Reflexive

SceneryRef As Reflexive

ScriptCrap As Reflexive

ScriptSyntaxDataSize As Integer

ScriptTriggers As Reflexive

SkyBox As Reflexive

SoundScenery As Reflexive

SoundSceneryRef As Reflexive

SourceFiles As Reflexive

StartingEquip As Reflexive

StartingLocations As Reflexive

StructBsp As Reflexive

TriggerVolumes As Reflexive

unk1 As Integer

unk2 As Integer

unk3 As Integer

Unknown1() As Reflexive

Unknown2 As Reflexive

Unknown3() As Reflexive

Unknown4 As Integer

Unknown5 As Reflexive

Unknown6() As Reflexive

Unknown7() As Integer

unk\_string1 As string

unk\_string2 As string

unk\_string3 As string

unneeded1() As Integer

unneeded2() As Integer

Vehicle As Reflexive

VehicleRef As Reflexive

VerifyCutscenes As Reflexive

VerifyCutsceneTitle As Reflexive

Weap As Reflexive

WeapRef As Reflexive

zero1() As Integer

End Class

## **Module Scenario Stuff**

### **Scenario\_Stuff.scenario\_header\_read:**

Function scenario\_header\_read(map as mapfile, index as integer) As Scenario\_Header

'So full of hate were our eyes

'That none of us could see

'Our war would yield countless dead

'But never victory  
,  
'So let us cast arms aside  
'And like discard our wrath  
'Thou, in faith, will keep us safe  
'Whilst we find the path

```
dim scnr_reader as binaryStream = map.f.openasbinaryFile
scnr_reader.LittleEndian = true
scnr_reader.Position = map.tags_list(index).offset //0 the first tag is usually the first tag
```

```
dim temp as new Reflexive
dim scenarioHDR1 as new Scenario_Header
scenarioHDR1.unk_string1 = scnr_reader.read(16)//16 characters
scenarioHDR1.unk_string2 = scnr_reader.read(16)//16 characters
scenarioHDR1.unk_string3 = scnr_reader.read(16)//16 characters
scenarioHDR1.SkyBox = temp.read(scnr_reader)
scenarioHDR1.unk1 = scnr_reader.readint32
scenarioHDR1.ChildScenarios = temp.read(scnr_reader)
for i as integer = 1 to 46
    scenarioHDR1.unneeded1.append( scnr_reader.readint32 )
next
scenarioHDR1.EditorScenarioSize = scnr_reader.readint32
scenarioHDR1.unk2 = scnr_reader.readint32
scenarioHDR1.unk3 = scnr_reader.ReadInt32
scenarioHDR1.pointertoindex = scnr_reader.ReadInt32
for i as integer = 1 to 2
    scenarioHDR1.unneeded2.append scnr_reader.ReadInt32
next
scenarioHDR1.pointertoendofindex = scnr_reader.Readint32
for i as integer = 1 to 57
    scenarioHDR1.zero1.append scnr_reader.readint32
next
scenarioHDR1.ObjectNames = temp.read(scnr_reader)
scenarioHDR1.Scenery = temp.read(scnr_reader)
scenarioHDR1.SceneryRef = temp.read(scnr_reader)
scenarioHDR1.Biped = temp.read(scnr_reader)
scenarioHDR1.BipedRef = temp.read(scnr_reader)
scenarioHDR1.Vehicle = temp.read(scnr_reader)
scenarioHDR1.VehicleRef = temp.read(scnr_reader)
scenarioHDR1.Equip = temp.read(scnr_reader)
scenarioHDR1.EquipRef = temp.read(scnr_reader)
scenarioHDR1.Weap = temp.read(scnr_reader)
scenarioHDR1.WeapRef = temp.read(scnr_reader)
scenarioHDR1.DeviceGroups = temp.read(scnr_reader)
scenarioHDR1.Machine = temp.read(scnr_reader)
scenarioHDR1.MachineRef = temp.read(scnr_reader)
scenarioHDR1.Control = temp.read(scnr_reader)
scenarioHDR1.ControlRef = temp.read(scnr_reader)
scenarioHDR1.LightFixture = temp.read(scnr_reader)
scenarioHDR1.LightFixtureRef = temp.read(scnr_reader)
scenarioHDR1.SoundScenery = temp.read(scnr_reader)
scenarioHDR1.SoundSceneryRef = temp.read(scnr_reader)
for i as integer = 1 to 7
```

```

    scenarioHDR1.Unknown1.append temp.read(scnr_reader)
next
scenarioHDR1.PlayerStartingProfile = temp.read(scnr_reader)
scenarioHDR1.PlayerSpawn = temp.read(scnr_reader)
scenarioHDR1.TriggerVolumes = temp.read(scnr_reader)
scenarioHDR1.Animations = temp.read(scnr_reader)
scenarioHDR1.MultiplayerFlags = temp.read(scnr_reader)
scenarioHDR1.MpEquip = temp.read(scnr_reader)
scenarioHDR1.StartingEquip = temp.read(scnr_reader)
scenarioHDR1.BspSwitchTrigger = temp.read(scnr_reader)
scenarioHDR1.Decals = temp.read(scnr_reader)
scenarioHDR1.DecalsRef = temp.read(scnr_reader)
scenarioHDR1.DetailObjCollRef = temp.read(scnr_reader)
for i as integer = 1 to 7
    scenarioHDR1.Unknown3.Append temp.read(scnr_reader)
next
scenarioHDR1.ActorVariantRef = temp.read(scnr_reader)
scenarioHDR1.Encounters = temp.read(scnr_reader)
//unconfirmed structs
scenarioHDR1.CommandLists = temp.read(scnr_reader)
scenarioHDR1.Unknown2 = temp.read(scnr_reader)
scenarioHDR1.StartingLocations = temp.read(scnr_reader)
scenarioHDR1.Platoons = temp.read(scnr_reader)
scenarioHDR1.AiConversations = temp.read(scnr_reader)
scenarioHDR1.ScriptSyntaxDataSize = scnr_reader.ReadInt32
scenarioHDR1.unknown4 = scnr_reader.readint32
scenarioHDR1.ScriptCrap = temp.read(scnr_reader)
scenarioHDR1.Commands = temp.read(scnr_reader)
scenarioHDR1.Points = temp.read(scnr_reader)
scenarioHDR1.AiAnimationRefs = temp.read(scnr_reader)
scenarioHDR1.GlobalsVerified = temp.read(scnr_reader)
scenarioHDR1.AiRecordingRefs = temp.read(scnr_reader)
scenarioHDR1.Unknown5 = temp.read(scnr_reader)
scenarioHDR1.Participants = temp.read(scnr_reader)
scenarioHDR1.Lines = temp.read(scnr_reader)
scenarioHDR1.ScriptTriggers = temp.read(scnr_reader)
scenarioHDR1.VerifyCutscenes = temp.read(scnr_reader)
scenarioHDR1.VerifyCutsceneTitle = temp.read(scnr_reader)
scenarioHDR1.SourceFiles = temp.read(scnr_reader)
scenarioHDR1.CutsceneFlags = temp.read(scnr_reader)
scenarioHDR1.CutsceneCameraPoi = temp.read(scnr_reader)
scenarioHDR1.CutsceneTitles = temp.read(scnr_reader)
for i as integer = 1 to 8
    scenarioHDR1.Unknown6.Append temp.read(scnr_reader)
next
for i as integer = 1 to 2
    scenarioHDR1.Unknown7.append scnr_reader.readint32
next
scenarioHDR1.StructBsp = temp.read(scnr_reader)

return scenarioHDR1
End Function
End Module

```

**STRUCT\_BIPED.dead:**

```
Sub dead(input as integer)
    dim temp as string
    temp = bin(input)
    dim max as integer = 32 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.mid(32, 1) = "1" then
        dead = true
    else
        dead = false
    end
End Sub
```

**STRUCT\_BIPED.not\_placed:**

```
Sub not_placed(input as integer)
    dim temp as string
    temp = bin(input)
    dim max as integer = 32 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.mid(32, 1) = "1" then
        not_placed_auto = true
    else
        not_placed_auto = false
    end
    if temp.mid(31, 1) = "1" then
        not_placed_easy = true
    else
        not_placed_easy = false
    end
    if temp.mid(30, 1) = "1" then
        not_placed_normal = true
    else
        not_placed_normal = false
    end
    if temp.mid(29, 1) = "1" then
        not_placed_hard = true
    else
        not_placed_hard = false
    end
End Sub
```

**STRUCT\_BIPED.readStruct:**

```
Sub readStruct(byref BR as binaryStream)
    br.LittleEndian = true
```

```

    position = br.Position

    type_index = br.ReadInt16
    name_index = br.ReadInt16
    dim not_placed_int as integer = br.ReadInt32
    not_placed(not_placed_int)

    x = br.ReadSingle
    y = br.ReadSingle
    z = br.ReadSingle
    yaw = br.ReadSingle //20
    pitch = br.ReadSingle //24
    roll = br.ReadSingle //28

    br.Position = position + &h48
    body_vitality = br.ReadShort

    br.Position = position + &h4C
    dim dead_int as integer = br.ReadInt32
    dead(dead_int)

    br.Position = position + 120
End Sub
body_vitality As Integer

dead As boolean

name As string

name_index As Integer

name_script As string

not_placed_auto As boolean

not_placed_easy As boolean

not_placed_hard As boolean

not_placed_normal As boolean

pitch As single

position As Integer

roll As single

type_index As Integer

x As single

y As single

yaw As single

```

z As single

End Class

## **Class STRUCT\_EQUIP**

numid As short

unk() As int32

unk2 As short

unk3 As int32

x As single

y As single

z As single

End Class

## **Class STRUCT\_VEHICLE\_SPAWN**

### **STRUCT\_VEHICLE\_SPAWN.get\_alive:**

```
Sub get_alive(input as integer)
    dim temp as string
    temp = bin(input)
    dim max as integer = 32 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.mid(32, 1) = "1" then
        alive = false
    else
        alive = true
    end
End Sub
```

### **STRUCT\_VEHICLE\_SPAWN.MP\_spawn\_bit\_read:**

```
Sub MP_spawn_bit_read(input as short)
    dim temp as string
    temp = bin(input)
    dim max as integer = 16 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.Mid(16,1) = "1" then
        MP_slayer_default = true
    end
End Sub
```

```

else
    MP_slayer_default = false
end
if temp.Mid(15,1) = "1" then
    MP_ctf_default = true
else
    MP_ctf_default = false
end
if temp.Mid(14,1) = "1" then
    MP_king_default = true
else
    MP_king_default = false
end
if temp.Mid(13,1) = "1" then
    MP_oddball_default = true
else
    MP_oddball_default = false
end
if temp.Mid(8,1) = "1" then
    MP_slayer_ok = true
else
    MP_slayer_ok = false
end
if temp.Mid(7,1) = "1" then
    MP_ctf_ok = true
else
    MP_ctf_ok = false
end
if temp.Mid(6,1) = "1" then
    MP_king_ok = true
else
    MP_king_ok = false
end
if temp.Mid(5,1) = "1" then
    MP_oddball_ok = true
else
    MP_oddball_ok = false
end
End Sub

```

### **STRUCT\_VEHICLE\_SPAWN.not\_placed:**

```

Sub not_placed(input as integer)
    dim temp as string
    temp = bin(input)
    dim max as integer = 32 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.mid(32, 1) = "1" then
        not_placed_auto = true
    else
        not_placed_auto = false
    end
end

```



```

if temp.mid(31, 1) = "1" then
    not_placed_easy = true
else
    not_placed_easy = false
end
if temp.mid(30, 1) = "1" then
    not_placed_normal = true
else
    not_placed_normal = false
end
if temp.mid(29, 1) = "1" then
    not_placed_hard = true
else
    not_placed_hard = false
end
End Sub

```

### **STRUCT\_VEHICLE\_SPAWN.readStruct:**

```

Sub readStruct(byref br as binaryStream)
    position = br.Position
    br.LittleEndian = true
    type = br.ReadShort
    name_index = br.ReadShort
    unknown1 = br.ReadInt32 //well sort of but ain't dealing with it now
    not_placed(unknown1)
    x = br.ReadSingle
    y = br.ReadSingle
    z = br.ReadSingle
    yaw = br.ReadSingle
    pitch = br.ReadSingle
    roll = br.ReadSingle

    br.Position = position + &h48
    body_vitality = br.ReadSingle
    dim alive as Integer = br.ReadInt32
    get_alive(alive)

    br.Position = position + &h58
    team_index = br.ReadInt8
    br.Position = position + &h5A
    dim MP_spawn_bitmask16 as short = br.ReadShort
    MP_spawn_bit_read(MP_spawn_bitmask16)

    br.Position = position + 120
End Sub
alive As boolean

body_vitality As single

MP_ctf_default As boolean

MP_ctf_ok As boolean

MP_king_default As boolean

```

MP\_king\_ok As boolean

MP\_oddball\_default As boolean

MP\_oddball\_ok As boolean

MP\_slayer\_default As boolean

MP\_slayer\_ok As boolean

### **STRUCT\_VEHICLE\_SPAWN.name:**

name As string

not in the normal struct

name\_index As short

not\_placed\_auto As boolean

not\_placed\_easy As boolean

not\_placed\_hard As boolean

not\_placed\_normal As boolean

pitch As single

position As Integer

roll As single

team\_index As int8

type As short

unknown1 As int32

x As single

y As single

yaw As single

z As single

End Class

## **Class STRUCT\_VEHICLE\_REF**

### **STRUCT\_VEHICLE\_REF.readStruct:**

Sub readStruct(byref br as binaryStream)

dim temp\_dep as new Dependency

```

    vehicle = temp_dep.read(br)

    redim unk(-1)
    for i as integer = 1 to 8
        unk.Append br.ReadInt32
    next
End Sub
unk(-1) As Integer

vehicle As dependency

End Class

```

## **Class STRUCT\_PALLETTE**

### **STRUCT\_PALLETTE.readStruct:**

```

Sub readStruct(byref br as binaryStream)
    br.LittleEndian = true
    position = br.Position

    tag = reverse(br.Read(4))
    NamePtr = br.ReadInt32
    zero = br.ReadInt32
    TagId = br.ReadInt32

    br.Position = position + 48
End Sub
NamePtr As int32

position As Integer

tag As string

TagId As int32

zero As Integer

End Class

```

## **Class STRUCT\_WEAPON\_SPAWN**

### **STRUCT\_WEAPON\_SPAWN.not\_placed:**

```

Sub not_placed(input as integer)
    dim temp as string
    temp = bin(input)
    dim max as integer = 32 - temp.len
    for i as integer = 1 to max
        temp = "0" + temp
    next

    if temp.mid(32, 1) = "1" then

```

```

        not_placed_auto = true
    else
        not_placed_auto = false
    end
    if temp.mid(31, 1) = "1" then
        not_placed_easy = true
    else
        not_placed_easy = false
    end
    if temp.mid(30, 1) = "1" then
        not_placed_normal = true
    else
        not_placed_normal = false
    end
    if temp.mid(29, 1) = "1" then
        not_placed_hard = true
    else
        not_placed_hard = false
    end
End Sub

```

### **STRUCT\_WEAPON\_SPAWN.readStruct:**

```

Sub readStruct(byref BR as binaryStream)
    br.LittleEndian = true
    position = br.Position

    type_index = br.ReadInt16
    name_index = br.ReadInt16
    dim not_placed_int as integer = br.ReadInt32
    not_placed(not_placed_int)

    x = br.ReadSingle
    y = br.ReadSingle
    z = br.ReadSingle
    yaw = br.ReadSingle
    pitch = br.ReadSingle
    roll = br.ReadSingle

    br.Position = position + &h48
    rounds_left = br.ReadShort
    rounds_loaded = br.ReadShort

    br.Position = position + 92
End Sub
name As string

name_index As Integer

name_script As string

not_placed_auto As boolean

not_placed_easy As boolean

```

not\_placed\_hard As boolean  
not\_placed\_normal As boolean  
pitch As single  
position As Integer  
roll As single  
rounds\_left As Integer  
rounds\_loaded As Integer  
type\_index As Integer  
x As single  
y As single  
yaw As single  
z As single  
End Class

## **Class STRUCT WEAPON\_REF**

flag As short  
numid As short  
rot As single  
unknown1 As int32  
unknown2() As int32  
x As single  
y As single  
z As single  
End Class

## **Class STRUCT\_SCENERY\_SPAWN**

### **STRUCT\_SCENERY\_SPAWN.not\_placed:**

```
Sub not_placed(input as integer)
    dim temp as string
    temp = bin(input)
```

```

dim max as integer = 32 - temp.len
for i as integer = 1 to max
    temp = "0" + temp
next

if temp.mid(32, 1) = "1" then
    not_placed_auto = true
else
    not_placed_auto = false
end
if temp.mid(31, 1) = "1" then
    not_placed_easy = true
else
    not_placed_easy = false
end
if temp.mid(30, 1) = "1" then
    not_placed_normal = true
else
    not_placed_normal = false
end
if temp.mid(29, 1) = "1" then
    not_placed_hard = true
else
    not_placed_hard = false
end
End Sub

```

### **STRUCT\_SCENERY\_SPAWN.readStruct:**

```

Sub readStruct(byref br as binaryStream)
    position = br.Position
    br.LittleEndian = true
    type_index = br.ReadInt16
    name_index = br.ReadInt16
    dim not_placed_int as integer = br.ReadInt32
    not_placed(not_placed_int)

    x = br.ReadSingle
    y = br.ReadSingle
    z = br.ReadSingle
    yaw = br.ReadSingle
    pitch = br.ReadSingle
    roll = br.ReadSingle

    br.Position = position + 72
End Sub

name As string

name_index As Integer

name_script As string

not_placed_auto As boolean

not_placed_easy As boolean

```

not\_placed\_hard As boolean  
not\_placed\_normal As boolean  
pitch As single  
position As Integer  
roll As single  
type\_index As Integer  
x As single  
y As single  
yaw As single  
z As single  
End Class

## **Class STRUCT\_MACHINE**

MachineType As short  
rot As single  
unk() As int32  
unk2 As short  
unk3 As int32  
x As single  
y As single  
z As single  
End Class

## **Class STRUCT\_SOUND\_SCENERY**

SoundType As short  
unk2 As short  
unk3 As int32  
unk4() As int32

x As single

y As single

z As single

End Class

## **Class STRUCT\_PLAYER\_SPAWN**

### **STRUCT\_PLAYER\_SPAWN.readStruct:**

Sub readStruct(byref br as binaryStream)

    br.LittleEndian = true

    position = br.Position

    x = br.ReadSingle

    y = br.ReadSingle

    z = br.ReadSingle

    rot = br.ReadSingle

    team\_index = br.ReadShort

    bsp\_index = br.ReadShort

    gametype\_enum1 = br.ReadShort

    gametype\_enum2 = br.ReadShort

    gametype\_enum3 = br.ReadShort

    gametype\_enum4 = br.ReadShort

    br.Position = position + 52

End Sub

bsp\_index As short

gametype\_enum1 As short

gametype\_enum2 As short

gametype\_enum3 As short

gametype\_enum4 As short

position As Integer

rot As single

team\_index As short

x As single

y As single

z As single

### **STRUCT\_PLAYER\_SPAWN Note: gametype enums**

gametype enums



0 – none  
1 – ctf  
2 – slayer  
3 – oddball  
4 – king of hill  
5 – race  
6 – terminator  
7 – stub  
8 – ignored1  
9 – ignored2  
10 – ignored3  
11 – ignored4  
12 – all games  
13 – all except CTF  
14 – all except CTF & race  
End Class

## **Class STRUCT\_PROFILE\_PLACEMENT**

fraggrenadecount As short

health As single

name As string

plasmagrencount As short

primaryclip As short

primaryidentifier As Integer

primaryrawfilename As Integer

primarytag As string

primarytotal As short

secondaryclip As short

secondaryfilename As Integer

secondaryidentifier As Integer

secondarytag As string

secondarytotal As short

shields As single

zero1 As string

zero2 As string

zero3 As string

End Class

## **Class STRUCT\_MP\_FLAG**

### **STRUCT\_MP\_FLAG.readStruct:**

Sub readStruct(byref br as binaryStream)

    position = br.Position

    br.LittleEndian = true

    x = br.ReadSingle

    y = br.ReadSingle

    z = br.ReadSingle

    yaw = br.ReadSingle

    type = br.ReadInt16

    teamIndex = br.ReadInt16

    select case type

    case 0

        name = "CTF flag"

    case 1

        name = "CTF vehicle"

    case 2

        name = "Oddball"

    case 3

        name = "Race track"

    case 4

        name = "Race vehicle"

    case 5

        name = "Vegas bank"

    case 6

        name = "Teleporter entrance"

    case 7

        name = "Teleporter exit"

    case 8

        name = "Hill"

    end select

    dim d as new Dependency

    dep = d.read(br)

    br.Position = position + 148

End Sub

dep As dependency

dep\_name As string

name As string

position As Integer

teamIndex As int16

### **STRUCT\_MP\_FLAG.type:**

type As int16

//enum16

ctf – flag – 0

ctf – vehicle – 1

oddbal – ball spawn – 2

race – track – 3

race – vehicle – 4

vegas – bank – 5

teleporter from – 6

teleporter to – 7

hill – flag – 8

x As single

y As single

yaw As single

z As single

End Class

## **Class STRUCT\_MP\_EQUIP**

### **STRUCT\_MP\_EQUIP.readstruct:**

Sub readstruct(byref br as binaryStream)

br.LittleEndian = true

position = br.Position

br.Position = position + 4

type0 = br.ReadInt16

type1 = br.ReadInt16

type2 = br.ReadInt16

type3 = br.ReadInt16

team\_index = br.ReadInt16

spawn\_time = br.ReadInt16

```
br.Position = position + &h40
x = br.ReadSingle
y = br.ReadSingle
z = br.ReadSingle
yaw = br.ReadSingle
```

```
dim temp as new Dependency
itmc = temp.read(br)
```

```
br.Position = position + 144
End Sub
dep_name As string
```

```
itmc As dependency
```

```
position As Integer
```

```
spawn_time As integer
```

```
team_index As Integer
```

```
type0 As Integer
```

```
type1 As Integer
```

```
type2 As Integer
```

```
type3 As Integer
```

```
x As single
```

```
y As single
```

```
yaw As single
```

```
z As single
```

```
End Class
```

## **Class STRUCT\_PLAYER\_STARTING\_PROFILE**

```
unk() As int32
```

```
End Class
```

## **Class STRUCT\_DEVICE\_GROUP**

```
unk As string
```

```
End Class
```

## **Class STRUCT\_BSP\_TRIGGER**

unk() As int32

End Class

## **Class STRUCT\_MOVE\_POSITIONS**

offset As int32

unk() As int32

unk1 As string

unk2 As string

End Class

## **Class STRUCT\_OBJECT\_NAMES**

### **STRUCT\_OBJECT\_NAMES.readStruct:**

Sub readStruct(byref br as binaryStream)

    br.LittleEndian = true

    position = br.Position

    dim tempint as int8 = br.ReadInt8

    dim tempstring as string = ""

    while tempint <> 0 AND br.Position < position + 32

        br.Position = br.Position - 1

        tempstring = tempstring + br.Read(1)

        tempint = br.ReadInt8

    wend

    name = tempstring

    br.Position = position + 36

End Sub

name As string

position As Integer

unk1 As short

unk2 As short

End Class

## **Class STRUCT\_TRIGGER\_VOLUMES**

box As BOUNDING\_BOX

name As string

unk As int32

unk2() As single

End Class

## **Class STRUCT\_ACTOR\_VARIANT\_REF**

NamePtr As int32

tag As string

unk As int32

unk2 As single

End Class

## **Class STRUCT\_AI\_ANIMATION\_REF**

name As string

unk() As int32

unk1 As short

unk2 As short

End Class

## **Class STRUCT\_AI\_SCRIPT\_REF**

name As string

unk() As int32

End Class

## **Class STRUCT\_AI\_RECORDING\_REF**

NamePtr As int32

tag As string

unk() As int32

unk1() As int32

End Class

Class STRUCT\_SCRIPT\_TRIGGER

name As string

unk() As int32

unk1 As int32

x As single

y As single

z As single

End Class

## **Class STRUCT\_GLOBALS**

name As string

unk() As int32

unk1 As int32

unk2 As int32

unk3() As single

x As single

y As single

z As single

End Class

## **Class STRUCT\_REFERENCES**

name As string

unk() As int32

unk1 As int32

End Class

## **Class STRUCT\_ENCOUNTER**

FiringPositions As Reflexive

Platoons As Reflexive

PlayerStartLocations As Reflexive

Squads As Reflexive

text As string

unk() As int32

End Class

## **Class STRUCT\_ENC\_SQUAD**

ActorType As short

Ai\_Attacking As int32

Ai\_AttackingGuard As int32

Ai\_AttackingSearch As int32

Ai\_Defending As int32

Ai\_DefendingGuard As int32

Ai\_DefendingSearch As int32

Ai\_Pursuing As int32

InitialState As Short

InsaneDiffCount As short

name As string

NormalDiffCount As short

Platoon As Short

ReturnState As Short

StartLocations As Reflexive

unk1() As int32

unk2() As int32

unk3() As int32

unk4() As int32

End Class

## **Class STRUCT\_ENC\_SQUAD\_SPAWN**



CommandList As short

unk2() As short

x As single

y As single

yaw As single

z As single

End Class

## **Class STRUCT\_ENCOUNTER\_INFO**

Squads() As STRUCT\_ENC\_SQUAD

SquadSpawns() As STRUCT\_ENC\_SQUAD\_SPAWN

End Class

## **Class STRUCT\_DECAL**

unk1 As short

unk2 As short

x As single

y As single

z As single

End Class

## **Class STRUCT\_DECAL\_REF**

NamePtr As int32

reserved As int32

tag As string

TagId As int32

End Class

## **Class STRUCT\_SHADER\_INDEX**

ShaderIndex As int32

ShaderType As int32

End Class

## **Class STRUCT\_SKYBOX**

NameRef As int32

tag As string

TagId As int32

unk1 As int32

End Class

## **Class STRUCT\_STARTING\_EQUIP**

unk() As int32

End Class

## **Class STRUCT\_CONTROL**

Tag\_Id As int32

unk() As int32

unk1 As short

unk2 As short

unk3 As short

x As single

y As single

z As single

End Class

## **Class STRUCT\_LIGHT\_FIXTURE**

unk() As int32

End Class

## **Class ustr\_edit\_control**

Inherits ContainerControl

### **ustr\_edit\_control.init:**

```
Sub init(in_f as folderitem, in_magic as integer, in_offset as integer)
    f = in_f
    magic = in_magic
    offset = in_offset

    refresh
End Sub
```

### **ustr\_edit\_control.refresh:**

```
Sub refresh()
    dim br as BinaryStream = f.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset

    dim string_reflexive as new reflexive
    dim reader as new reflexive
    string_reflexive = reader.read(br)

    br.Position = string_reflexive.offset - magic
    redim strings(-1)
    for i as integer = 1 to string_reflexive.count
        dim temp_snippet as new ustr_snippet
        temp_snippet.init(br, magic)
        strings.Append temp_snippet
    next

    for i as integer = 1 to string_reflexive.count
        strings(i-1).read(br)
    next

    dim selected as integer
    if ustr_selected then
        selected = strings_list.ListIndex
    end
    strings_list.DeleteAllRows
    for i as integer = 0 to UBound(strings)
        strings_list.addrow(strings(i).myString)
    next
    change_ok = false
    if ustr_selected then
        strings_list.listindex = selected
        selected_ustr = strings(selected)
        EditField1.text = selected_ustr.myString
    else
        EditField1.text = ""
    end
    change_ok = true

    br.close
End Sub
```

ustr\_edit\_control.write:

Sub write()

    //do something

    if not ustr\_selected then return

    dim bw as BinaryStream = f.OpenAsBinaryFile(true)

    dim len\_dif as integer = (selected\_ustr.maxLength/2) - selected\_ustr.myString.len

    bw.LittleEndian = true

    bw.Position = selected\_ustr.offset

    //cap off any additional writing

    selected\_ustr.myString = selected\_ustr.myString.left(selected\_ustr.maxLength/2)

    if len\_dif > 0 then

        for i as integer = 1 to len\_dif

            selected\_ustr.myString = selected\_ustr.myString + " "

        next

    end

    for i as integer = 1 to selected\_ustr.myString.len

        bw.Write(selected\_ustr.myString.mid(i,1))

        bw.Writeuint8(0)

    next

    bw.close

    refresh

End Sub

change\_ok As boolean

f As folderItem

magic As Integer

offset As integer

selected\_ustr As ustr\_snippet

strings() As ustr\_snippet

ustr\_selected As boolean

### **ustr\_edit\_control Control strings\_list:**

Function CellClick(row as Integer, column as Integer, x as Integer, y as Integer) As Boolean

    change\_ok = false

    selected\_ustr = strings(row)

    ustr\_selected = true

    EditField1.text = selected\_ustr.myString

    change\_ok = true

End Function

Function KeyDown(Key As String) As Boolean

    dim row as integer

```

dim ok as boolean = false
select case asc(key)
case 31
    if strings_list.ListIndex + 1 < strings_list.listcount then
        row = strings_list.ListIndex + 1
        ok = true
    end
case 30
    if strings_list.ListIndex - 1 > -1 then
        row = strings_list.ListIndex - 1
        ok = true
    end
end

if ok then
    change_ok = false
    selected_ustr = strings(row)
    ustr_selected = true
    EditField1.text = selected_ustr.myString
    change_ok = true
end
End Function

```

### **ustr\_edit\_control Control EditField1:**

```

Function KeyDown(Key As String) As Boolean
    if asc(key) = 3 or asc(key) = 13 then
        if not change_ok then return true
        if not ustr_selected then return true
        selected_ustr.myString = EditField1.text
        write
        return true
    end
End Function
End Class

```

## **Class ustr\_snippit**

### **ustr\_snippit.init:**

```

Sub init(byref br as binarystream, magic as integer)
    br.LittleEndian = true

    maxLength = br.readint32
    zero1 = br.readint32
    old_offset = br.readint32
    offset = br.readint32 - magic
    zero2 = br.readint32
End Sub

```

### **ustr\_snippit.read:**

```

Sub read(byref br as binarystream)
    br.LittleEndian = true
    br.Position = offset

```

```

mystring = ""
for i as integer = 1 to maxLength
    dim temp as uint8 = br.readuint8
    if temp <> 0 then
        mystring = mystring + chr(temp)
    end
next
End Sub
maxLength As Integer

```

myString As string

offset As Integer

### **ustr\_snippit.old\_offset:**

```

old_offset As Integer
//would be the internal offset but delimiters make it inaccurate
zero1 As Integer

```

zero2 As Integer

End Class

## **Class HexEditControl**

Inherits ContainerControl

### **HexEditControl.OpenFile:**

```

Sub OpenFile(aFile as folderItem)
    dim bs as BinaryStream
    dim data as String

    if aFile <> nil and aFile.Length > 0 then
        bs = aFile.OpenAsBinaryFile(False)
        if bs <> nil then
            mSData = bs.Read(bs.Length)
            bs.Close

            HexEditCanvas1.Refresh

        end if
    end if
End Sub
Protected mQuietScroll As boolean

```

Protected mSData As string

### **HexEditControl Control HexEditCanvas1:**

```

Sub MouseExit()
    ByteField.Text = ""

```

End Sub

Sub MouseMove(X As Integer, Y As Integer)

ByteField.text = Format(HexEditCanvas1.PointToByteOffset(x,y), "#")

End Sub

Function GetFontSize() As integer

Return CDbI(FontSizePopup.List(FontSizePopup.ListIndex))

End Function

Function GetFont() As string

Return "Andale Mono"

if FontsPopup.ListIndex > -1 then

Return FontsPopup.List(FontsPopup.ListIndex)

end if

Return ""

End Function

Sub DeleteData(aLen as integer, aPosition as integer)

mSData = leftb(mSData, aPosition) + midb(mSData,aPosition+aLen+1)

End Sub

Sub ModifyData(aNewData as string, aPosition as integer, aOldLength as integer)

mSData = leftb(mSData, aPosition) + aNewData + midb(mSData,aPosition+aOldLength+1)

End Sub

Sub SetScrollPosition(aPos as integer)

mQuietScroll = true

HexBar.Value = aPos - 32767

mQuietScroll = False

End Sub

Sub DropObject(obj As DragItem, action As Integer)

if obj.FolderItemAvailable then

OpenFile obj.FolderItem

end if

End Sub

Sub Open()

me.AcceptFileDrop "special/any"

me.SetDrawStripes DrawStripesBox.Value

End Sub

Sub SetScrollPageIncrement(alncrement as integer)

mQuietScroll = true

HexBar.PageStep = alncrement

mQuietScroll = False

End Sub

Sub SetScrollMaximum(aMax as integer)

mQuietScroll = true

HexBar.Maximum = aMax - 32767

HexBar.Minimum = -32767

```
    mQuietScroll = False
End Sub
```

```
Function GetTotalDataLength() As integer
    Return lenb(mSData)
End Function
```

```
Function GetData(aLen as integer, aPosition as integer) As string
    Return midb(mSData, aPosition+1, aLen)
End Function
```

```
Sub InsertData(aData as string, aLoc as integer)
    mSData = leftb(mSData, aLoc) + aData + midb(mSData,aLoc+1)
End Sub
```

### **HexEditControl Control DrawStripesBox:**

```
Sub Action()
    HexEditCanvas1.SetDrawStripes me.Value
    HexEditCanvas1.Refresh
End Sub
```

### **HexEditControl Control HexBar:**

```
Sub ValueChanged()
    if not mQuietScroll then
        HexEditCanvas1.SetTopLine(me.Value + 32767)
        HexEditCanvas1.Refresh
    end if
End Sub
```

### **HexEditControl Control Timer1:**

```
Sub Action()
    HexEditCanvas1.BlinkInsertion
End Sub
```

### **HexEditControl Control FontsPopup:**

```
Sub Open()
    dim fnts(-1) as String
    dim i as Integer

    ReasonableFonts(fnts)
    if UBound(fnts) > -1 then
        for i = 0 to ubound(fnts)
            me.AddRow fnts(i)
        next
    end if
    me.ListIndex = 0
End Sub
```

```
Sub Change()
    HexEditCanvas1.Refresh
End Sub
```

### **HexEditControl Control FontSizePopup:**



```
Sub Change()  
    HexEditCanvas1.Refresh  
End Sub
```

### **HexEditControl Control PopupMenu1:**

```
Sub Change()  
    Select case me.ListIndex  
    case 0  
        HexEditCanvas1.SetNumberOfCharactersForOffset 0  
    case 1  
        HexEditCanvas1.SetNumberOfCharactersForOffset -1  
    case 2  
        HexEditCanvas1.SetNumberOfCharactersForOffset 4  
    case 3  
        HexEditCanvas1.SetNumberOfCharactersForOffset 8  
    end Select  
    HexEditCanvas1.Refresh  
End Sub
```

### **HexEditControl Control ShowTextBox:**

```
Sub Action()  
    HexEditCanvas1.SetVisibilityOfText me.Value  
    HexEditCanvas1.Refresh  
End Sub  
End Class
```

## **Class HexEditCanvas**

Inherits Canvas

```
Event DeleteData(aLen as integer, aPosition as integer)  
Sub ()  
Event GetData(aLen as integer, aPosition as integer) As string  
Sub ()  
Event GetFont() As string  
Sub ()  
Event GetFontSize() As integer  
Sub ()  
Event GetTotalDataLength() As integer  
Sub ()  
Event InsertData(aData as string, aLoc as integer)  
Sub ()  
Event KeyDown(Key as string) As boolean  
Sub ()  
Event ModifyData(aNewData as string, aPosition as integer, aOldLength as integer)  
Sub ()  
Event MouseDown(X as integer, Y as integer) As boolean  
Sub ()  
Event MouseEnter()  
Sub ()  
Event Open()  
Sub ()  
Event SetScrollMaximum(aMax as integer)
```

```

Sub ()
Event SetScrollPageIncrement(aIncrement as integer)
Sub ()
Event SetScrollPosition(aPos as integer)
Sub ()

```

### **HexEditCanvas.KeyDown:**

```

Function KeyDown(Key As String) As Boolean
    dim code as Integer
    dim res as Boolean

    res = KeyDown(Key)
    if res then
        Return True
    end if

    if Keyboard.CommandKey then
        Return False
    end if

    code = asc(Key)

    if code >= kArrowLeft and code <= kArrowDown Then
        HandleKeyMovement(code, Keyboard.ShiftKey)
        Return True
    end if

    if not HandleTyping(key, False) then
        Beep
    end if

    Return true
End Function

```

### **HexEditCanvas.MouseDown:**

```

Function MouseDown(X As Integer, Y As Integer) As Boolean
    dim curr as Integer
    dim res as Boolean

    res = MouseDown(X, Y)
    if res then
        Return res
    end if
    PaintInsertion(me.Graphics, True)
    if PointInText(x,y) then
        mTextIsFocus = true
    else
        mTextIsFocus = False
    end if

    if Keyboard.ShiftKey then
        curr = PointToByteOffset(x,y)
        if curr > mSelStart then
            mClickStartOffset = mSelStart

```

```

    else
        mClickStartOffset = mSelStart + mSelLength
    end if
else
    mClickStartOffset = PointToByteOffset(x,y)
    mSelStart = mClickStartOffset
    mSelLength = 0
end if
Return True
End Function

```

### HexEditCanvas.MouseDrag:

```

Sub MouseDrag(X As Integer, Y As Integer)
    dim current as Integer
    dim lineOffset as Integer
    dim newX, newY as Integer

    if y < 0 or y > me.Height then
        if y < 0 then
            lineOffset = -1
        else
            lineOffset = 1
        end if
        me.SetTopLine GetTopLine + lineOffset
        SetScrollPosition GetTopLine
        if y < 0 then
            newY = 1
            newX = HexLeftSide
        else
            newY = (me.Height \ mXHeight) * mXHeight - 2
            newX = me.Width
        end if
    else
        newY = y
        newX = x
    end if

    current = PointToByteOffset(newx,newy)
    if current < mClickStartOffset then
        mSelStart = current
        mSelLength = mClickStartOffset - mSelStart
        mSelDirection = -1
    else
        mSelStart = mClickStartOffset
        mSelLength = current - mClickStartOffset
        mSelDirection = 1
    end if
    me.Refresh

End Sub

```

### HexEditCanvas.MouseEnter:

```

Sub MouseEnter()
    if mWriteable then

```

```

        self.MouseCursor = IBeamCursor
    else
        self.MouseCursor = ArrowCursor
    end if
    MouseEnter
End Sub

```

### HexEditCanvas.Open:

```

Sub Open()
    mBytesPerGroup = 2
    mShowText = true
    SetDisplayCharacteristics(me.Graphics)
    CacheCommonData(me.Graphics)
    mCharactersForOffset = -1
    mSelStart = 0
    mSelLength = 0
    mWriteable = true
    Open
End Sub

```

### HexEditCanvas.Paint:

```

Sub Paint(g As Graphics)
    'g.DrawString GetData(500,0), 0,0
    dim x,y as Integer
    SetDisplayCharacteristics(g)
    'SanityChecks
    CalculateDisplayedText

    PaintBackground(g)
    PaintSelection(g)
    g.ForeColor = mTextColor

    if mCharactersForOffset <> 0 then
        g.DrawString mOffsetsAsDisplayed,kLeftMargin,mXHeight
        x = LeftHandWidth - (kLeftExtraSpace/2)
        g.ForeColor = mDividerColor
        g.DrawLine x, 0, x, g.Height
        g.ForeColor = mTextColor
    end if
    g.DrawString mDataAsDisplayed, LeftHandWidth, mXHeight
    if mShowText then
        x = g.Width-(kRightMargin + mBytesPerLine * mMWidth)
        g.DrawString mTextAsDisplayed, x, mXHeight
        x = x - (mMWidth/2)
        g.ForeColor = mDividerColor
        g.DrawLine x, 0, x, g.Height
        g.ForeColor = mTextColor
    end if
End Sub
Sub Activate()

End Sub

```

### **HexEditCanvas.BlinkInsertion:**

```
Sub BlinkInsertion()  
    PaintInsertion(me.Graphics, mInsertionOn)  
End Sub
```

### **HexEditCanvas.BytelsVisible:**

```
Function BytelsVisible(aOffset as Integer) As Boolean  
    Return aOffset >= mFirstDisplayedOffset and aOffset < (mFirstDisplayedOffset + LinesDisplayed*mBytesPerLine)  
End Function
```

### **HexEditCanvas.CacheCommonData:**

```
Protected Sub CacheCommonData(g as Graphics)  
    'dim g as Graphics  
    'g = me.Graphics  
    'SetDisplayCharacteristics(g)  
    mXHeight =  
    g.StringHeight("X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+"X"+chr(13)+chr(13)+"X"+chr(13)+"X"+chr(13),65536)/10  
    mMWidth = g.StringWidth("MMMMMMMMMMMM")/10  
    mBackColor = rgb(255,255,255)  
    mTextColor = rgb(0,0,0)  
    mDividerColor = rgb(160,160,160)  
    mStripeColor = RGB(237,243,254)  
End Sub
```

### **HexEditCanvas.CalculateDisplayedText:**

```
Protected Sub CalculateDisplayedText()  
    dim leftWidth as Integer  
    dim rightWidth as Integer  
    dim interslop as Integer  
    dim availableWidth as Integer  
    dim g as Graphics  
    dim charSlots as Integer  
    dim slotsPerGroup as Integer  
    dim data as String  
    dim lines as Integer  
    dim i as Integer  
    dim work as String  
    dim savedWork as String  
    dim interim as String  
    dim b as Integer  
    dim line as String  
    dim re as RegEx  
  
    g = me.Graphics  
    SetDisplayCharacteristics(g)  
    leftWidth = LeftHandWidth  
    rightWidth = kRightMargin  
    if mShowText then  
        interslop = kInterTextSlop  
    end if
```

```

availableWidth = g.Width - (leftWidth+rightWidth+interslop)
charSlots = availableWidth / mMWidth
slotsPerGroup = (mBytesPerGroup*2)+1
if mShowText then
    slotsPerGroup = slotsPerGroup+mBytesPerGroup
end if
mBytesPerLine = (charSlots\slotsPerGroup)* mBytesPerGroup

```

SanityChecks

```

lines = LinesDisplayed
data = GetData( mBytesPerLine*lines, mFirstDisplayedOffset)

```

```

SetScrollMaximum max(0, (ceil(GetTotalDataLength()/mBytesPerLine))-lines)
SetScrollPageIncrement lines-1
SetScrollPosition GetTopLine

```

```

mOffsetsAsDisplayed = ""
mTextAsDisplayed = ""
mDataAsDisplayed= ""
for i = 0 to lines
    mOffsetsAsDisplayed = mOffsetsAsDisplayed + OffsetTextForPosition(mFirstDisplayedOffset +
(i*mBytesPerLine), GetOffsetCharLength) + chr(13)
    work = left(data, mBytesPerLine)
    savedWork = work
    data = mid(data, mBytesPerLine+1)

```

```

    line = ""
    for b = ((mBytesPerLine/mBytesPerGroup)-1) DownTo 0
        'mBytesPerLine-mBytesPerGroup Step mBytesPerGroup
        interim = leftb(work, mBytesPerGroup)
        work = midb(work, mBytesPerGroup+1)
        line = line + DataToHex(interim) + " "
    next
    mDataAsDisplayed = mDataAsDisplayed + line + chr(13)

```

```

if mShowText then
    'work = ReplaceAll(savedWork, chr(13), ".")
    'work = ReplaceAll(work, chr(10), ".")
    'work = ReplaceAll(work, chr(0), ".")
    're = new RegEx
    're.SearchPattern = "["+mBadChars+"]"
    're.ReplacementPattern = "\"."
    're.Options.ReplaceAllMatches = true
    'work = re.Replace(savedWork)
    work = mBadCharsRE.Replace(savedWork)

    mTextAsDisplayed = mTextAsDisplayed + work + chr(13)
end if
next

```

End Sub

**HexEditCanvas.DataToHex:**

Protected Function DataToHex(aString as string) As string

```
dim i as Integer
dim res as String
dim data as String
```

```
data = aString
for i = 0 to lenb(aString)-1
    res = res + right("0"+hex(ascb(data)), 2)
    data = midb(data, 2)
next
Return res
```

End Function

Sub Deactivate()

End Sub

### HexEditCanvas.DetermineBadCharacters:

Protected Sub DetermineBadCharacters(aFont as string, aSize as integer, g as graphics)

```
dim i as Integer
'dim g as Graphics
dim cWidth as Double
dim mWidth as Double
```

```
'g = me.Graphics
mBadChars = "\000\x0A\x0D"
```

```
mWidth = g.StringWidth("MMMMMMMMMM") / 10
```

```
for i = 1 to 255
```

```
    if i <> 10 and i <> 13 then
```

```
        cWidth = g.StringWidth(chrb(i) + chrb(i) + chrb(i) + chrb(i) + chrb(i) + chrb(i) + chrb(i) + chrb(i) + chrb(i) +
chrb(i)) / 10
```

```
        if cWidth <> mWidth then
```

```
            mBadChars = mBadChars + chrb(i)
```

```
        end if
```

```
    end if
```

```
Next
```

```
mBadCharsRE = new RegEx
```

```
mBadCharsRE.SearchPattern = "["+mBadChars+"]"
```

```
mBadCharsRE.ReplacementPattern = "\""
```

```
mBadCharsRE.Options.ReplaceAllMatches = true
```

```
mBadCharsRE.Options.CaseSensitive = False
```

```
'mBadCharsRE.Options.UTF8 = False
```

End Sub

### HexEditCanvas.GetOffsetCharLength:

Protected Function GetOffsetCharLength() As integer

```
if mCharactersForOffset > -1 Then
```

```
    Return mCharactersForOffset
```

```
end if
```

```
return max(len(hex(GetTotalDataLength())), len(hex(LinesDisplayed*mBytesPerLine)))
```

End Function

### HexEditCanvas.GetTopLine:

Function GetTopLine() As Integer

Return mFirstDisplayedOffset / mBytesPerLine

End Function

### HexEditCanvas.HandleKeyMovement:

Sub HandleKeyMovement(aKey as Integer, aExtendSelection as boolean)

dim newStart as Integer

dim newLen as Integer

dim oldLen, oldStart as Integer

if mInsertionOn then

PaintInsertion(me.Graphics, true)

end if

if not aExtendSelection then

if mSelLength > 0 then

oldLen = mSelLength

oldStart = mSelStart

end if

Select case aKey

case kArrowLeft

newStart = mSelStart-1

case kArrowRight

newStart = (mSelStart+mSelLength) + 1

case kArrowUp

newStart = mSelStart - mBytesPerLine

case kArrowDown

newStart = (mSelStart+mSelLength) + mBytesPerLine

end Select

if newStart < 0 then

newStart = 0

elseif newStart > GetTotalDataLength() then

newStart = GetTotalDataLength()

end if

if mSelStart = newStart and mSelLength = 0 then

Beep

end if

mSelStart = newStart

mSelLength = 0

if oldLen > 0 then

UpdateByteRange(oldStart, oldLen)

end if

RevealByte(mSelStart)

PaintInsertion(me.Graphics, False)

else

newStart = mSelStart

newLen = mSelLength

if mSelLength = 0 then

if aKey = kArrowLeft or aKey = kArrowUp then

mSelDirection = -1

else

mSelDirection = 1



```

    end if
end if
select case aKey
case kArrowLeft
    if mSelDirection < 0 then
        newStart = mSelStart - 1
        newLen = mSelLength + 1
    else
        newLen = mSelLength - 1
    end if
case kArrowRight
    if mSelDirection < 0 then
        newStart = mSelStart + 1
        newLen = mSelLength - 1
    else
        newLen = mSelLength + 1
    end if
case kArrowDown
    if mSelDirection < 0 then
        newStart = mSelStart + mBytesPerLine
        newLen = mSelLength - mBytesPerLine

    else
        newLen = mSelLength + mBytesPerLine
    end if
case kArrowUp
    if mSelDirection < 0 then
        newStart = mSelStart - mBytesPerLine
        newLen = mSelLength + mBytesPerLine
    else
        newLen = mSelLength - mBytesPerLine
    end if
end Select
if newLen < 0 then
    newStart = newStart + newLen
    newLen = abs(newLen)
    mSelDirection = -mSelDirection
    if mSelDirection = 0 then
        mSelDirection = -1
    end if
end if
if newStart < 0 then
    newLen = newLen + newStart 'newStart is neg so add it to subtract it
    newStart = 0
elseif newStart > GetTotalDataLength() then
    newLen = newLen - (newStart - GetTotalDataLength())
    newStart = GetTotalDataLength()
elseif newStart + newLen > GetTotalDataLength() then
    newLen = GetTotalDataLength() - newStart
end if
if mSelStart = newStart and mSelLength = newLen then
    Beep
end if
oldStart = mSelStart

```

```

oldLen = mSelLength
mSelStart = newStart
mSelLength = newLen
if mSelDirection < 0 then
    RevealByte(mSelStart)
else
    RevealByte(mSelStart+mSelLength)
end if
UpdateByteRange(min(mSelStart, oldStart), max(mSelLength, oldLen))
end if

```

End Sub

### HexEditCanvas.HandleTyping:

Function HandleTyping(aText as string, aForceBinary as boolean) As boolean

```

dim data as String
dim code as Integer
dim oldData as String

if not mWriteable then

    Return False
end if
if lenb(aText) = 1 and ascb(aText) = kDeleteKey then
    if mSelLength > 0 then
        DeleteData mSelLength, mSelStart
        mSelLength = 0
    elseif mSelStart > 0 then
        DeleteData 1, mSelStart-1
        mSelStart = mSelStart - 1
    else
        Return False
    end if
    RevealByte(mSelStart)
    me.UpdateByteRange mSelStart, &h7FFFFFFF
    mOnLowNibble = False

    Return True
end if

if TextIsFocus or aForceBinary then
    InsertText(aText)
    mOnLowNibble = False
    mSelLength = 0
    me.UpdateByteRange mSelStart, &h7FFFFFFF
    mSelStart = mSelStart+lenb(aText)
    mLastInsertion = mSelStart
    Return true
end if

if len(aText) < 1 then
    Return False
end if

```

```

if len(aText) = 1 then
    'handle special case of typing vs. pasting
    code = Asc(aText)
    if (code >= kZeroCode and code <= kNineCode) or (code >= kACode and code <= kFCode) or (code >=
    kLACode and code <= kLFCode) then

        else
            Return False
        end if
    if mLastInsertion <> mSelStart then
        mOnLowNibble = False
    end if
    if not mOnLowNibble then
        data = chrb(16*CDBl("&h"+aText))
    else
        data = chrb(CDBl("&h"+aText))
    end if

else
    data = HECCConvertTextToBinary(aText)
    mOnLowNibble = False
end if
if data = "" then
    Return False
end if

if mSelLength > 0 then
    ModifyData data, mSelStart, mSelLength
    mOnLowNibble = lenb(data) = 1
else
    if mOnLowNibble and lenb(data) = 1 then
        oldData = GetData(1, mSelStart-1)
        data = chrb(ascb(data) + ascb(oldData))
        ModifyData data, mSelStart-1, 1
        data = ""
        mOnLowNibble = False
    else
        InsertData data, mSelStart
        mOnLowNibble = lenb(data) = 1
    end if
end if

mSelLength = 0
mSelStart = mSelStart + lenb(data)
me.UpdateByteRange mSelStart-lenb(data), &h7FFFFFFF
mLastInsertion = mSelStart
Return true
End Function

```

### HexEditCanvas.HasSelection:

```

Function HasSelection() As boolean
    Return mSelLength > 0
End Function

```

**HexEditCanvas.HexAreaWidth:**

Function HexAreaWidth() As integer

if mShowText then

Return TextLeftSide - HexLeftSide - mMWidth - kInterTextSlop

end if

Return me.Width - HexLeftSide

End Function

**HexEditCanvas.HexIsFocus:**

Function HexIsFocus() As boolean

Return not TextIsFocus

End Function

**HexEditCanvas.HexLeftSide:**

Function HexLeftSide() As integer

Return LeftHandWidth

End Function

**HexEditCanvas.InsertText:**

Protected Sub InsertText(aText as string)

if mSelLength > 0 then

ModifyData aText, mSelStart, mSelLength

else

InsertData aText, mSelStart

end if

'me.UpdateByteRange mSelStart, &h7FFFFFFF

'mSelStart = mSelStart + lenb(aText)

End Sub

**HexEditCanvas.LeftHandWidth:**

Function LeftHandWidth() As integer

dim chars as Integer

chars = GetOffsetCharLength

if chars < 1 then

Return kLeftMargin

end if

Return kLeftMargin + kLeftExtraSpace + mMWidth\*chars

End Function

**HexEditCanvas.LinesDisplayed:**

Function LinesDisplayed() As integer

Return me.Height\mXHeight

End Function

**HexEditCanvas.OffsetLeftSide:**

Function OffsetLeftSide() As integer

Return kLeftMargin

End Function

HexEditCanvas.OffsetTextForPosition:

Protected Function OffsetTextForPosition(aOffset as Integer, aCharsPerOffset as Integer) As String

Return right("00000000"+Hex(aOffset), aCharsPerOffset)

End Function

### HexEditCanvas.PaintBackground:

Protected Sub PaintBackground(g as Graphics)

dim i as Integer

'dim w as Double

g.ForeColor = mBackColor

g.FillRect 0,0,me.Width, me.Height

if mDrawStripes then

g.ForeColor = mStripeColor

for i = 0 to LinesDisplayed-1 Step 2

g.FillRect 0, i\*mXHeight, g.Width, mXHeight

next

end if

'g.ForeColor = rgb(0,0,0)

'w = HexLeftSide

'do

'g.DrawLine w,0, w, me.Height

'w = w + mMWidth

'loop until w >= TextLeftSide

End Sub

### HexEditCanvas.PaintInsertion:

Sub PaintInsertion(g as Graphics, aErase as Boolean)

dim line as Integer

dim offset as Integer

dim x, y as Integer

if mWritable and mSelLength = 0 then

if not BytelsVisible(mSelStart) then

Return

end if

line = (mSelStart - mFirstDisplayedOffset)\mBytesPerLine

offset = (mSelStart - mFirstDisplayedOffset) mod mBytesPerLine

y = line \* mXHeight

if HexIsFocus then

x = (offset \ mBytesPerGroup) \* (mMWidth \* (mBytesPerGroup\*2 + 1))

x = x + (offset mod mBytesPerGroup)\*mMWidth\*2 //2 'cause 2 chars = 1 byte

x = x + LeftHandWidth

else

x = TextLeftSide + mMWidth\*offset

end if

if aErase then

```

    if line mod 2 = 0 and mDrawStripes then
        g.ForeColor = mStripeColor
    else
        g.ForeColor = rgb(255,255,255)
    end if
    mInsertionOn = False
else
    g.ForeColor = rgb(0,0,0)
    mInsertionOn = true
end if
g.DrawLine x-1, y+2, x-1, y+mXHeight
end if
End Sub

```

### HexEditCanvas.PaintSelection:

```

Protected Sub PaintSelection(g as graphics)
    dim startLine, startColumn as Integer
    dim endLine, endColumn as Integer
    dim offset as Integer
    dim startByteFromLine, endByteFromLine as Integer
    dim firstLine, middleLines, lastLine as RectShape
    dim textFirstLine, textMiddleLines, textLastLine as RectShape
    dim c as Color
    dim fig as FigureShape

    if mSelLength = 0 then
        PaintInsertion(g, true)
    else
        if mSelStart >= mFirstDisplayedOffset + (mBytesPerLine * LinesDisplayed) then
            Return
        end if
        if mSelStart + mSelLength < mFirstDisplayedOffset then
            Return
        end if
        'g.ForeColor = rgb(0,75, 255)
        offset = mSelStart - mFirstDisplayedOffset
        if offset < 0 then
            offset = 0
        end if
        startLine = offset / mBytesPerLine
        startByteFromLine = (offset mod mBytesPerLine)
        startColumn = (startByteFromLine \ mBytesPerGroup) * (mBytesPerGroup*2+1) + (startByteFromLine mod mBytesPerGroup)*2
        'g.FillRect startColumn*mMWidth+LeftHandWidth, startLine*mXHeight, mMWidth, mXHeight

        offset = (mSelStart + mSelLength) - mFirstDisplayedOffset
        if offset > LinesDisplayed * mBytesPerLine then
            offset = LinesDisplayed * mBytesPerLine
        end if
        endLine = offset / mBytesPerLine
        endByteFromLine = (offset mod mBytesPerLine)
        endColumn = (endByteFromLine \ mBytesPerGroup) * (mBytesPerGroup*2+1) + (endByteFromLine mod mBytesPerGroup)*2
    end if
end Sub

```

```

'if startLine < GetTopLine then
'startLine = GetTopLine
'startColumn = 0
'end if
,

'if endLine > GetTopLine + LinesDisplayed then
'endLine = GetTopLine + LinesDisplayed
'endColumn = (mBytesPerLine \ mBytesPerGroup) * (mBytesPerGroup*2+1) + (mBytesPerLine mod
mBytesPerGroup)*2
'end if
,

```

```

if endLine < startLine then
    Return
end if
if endLine <> startLine then
    firstLine = new RectShape
    firstLine.Width = HexAreaWidth - (startColumn*mMWidth)
    firstLine.Height = mXHeight
    firstLine.x = HexLeftSide+startColumn*mMWidth //+firstLine.Width/2
    firstLine.y = startLine*mXHeight //+ firstLine.Height/2

    lastLine = new RectShape
    lastLine.Height = mXHeight
    lastLine.Width = endColumn * mMWidth
    lastLine.x = HexLeftSide //+lastLine.Width/2
    lastLine.y = endLine*mXHeight //+ lastLine.Height/2
    if endLine > startLine + 1 then
        'create middle lines
        middleLines = new RectShape
        middleLines.Width = HexAreaWidth
        middleLines.Height = (endLine-startLine-1)*mXHeight
        middleLines.x = HexLeftSide //+middleLines.Width/2
        middleLines.y = (startLine+1)*mXHeight //+ middleLines.Height/2
    end if
else
    firstLine = new RectShape
    firstLine.Height = mXHeight
    firstLine.Width = (endColumn - startColumn) * mMWidth
    firstLine.x = HexLeftSide+startColumn*mMWidth //+firstLine.Width/2
    firstLine.y = startLine*mXHeight //+ firstLine.Height/2
end if

```

```

'Now calculate for the text area, if needed
if VisibilityOfText = true then
    if endLine <> startLine then
        textFirstLine = new RectShape
        textFirstLine.Width = TextAreaWidth - (startByteFromLine * mMWidth)
        textFirstLine.Height = mXHeight
        textFirstLine.x = TextLeftSide+startByteFromLine*mMWidth
        textFirstLine.y = startLine*mXHeight //+ firstLine.Height/2

        textLastLine = new RectShape
    end if
end if

```

```

textLastLine.Height = mXHeight
textLastLine.Width = endByteFromLine * mMWidth
textLastLine.x = TextLeftSide //+lastLine.Width/2
textLastLine.y = endLine*mXHeight //+ lastLine.Height/2
if endLine > startLine + 1 then
    'create middle lines
    textMiddleLines = new RectShape
    textMiddleLines.Width = TextAreaWidth
    textMiddleLines.Height = (endLine-startLine-1)*mXHeight
    textMiddleLines.x = TextLeftSide //+middleLines.Width/2
    textMiddleLines.y = (startLine+1)*mXHeight //+ middleLines.Height/2
end if
else
    textFirstLine = new RectShape
    textFirstLine.Height = mXHeight
    textFirstLine.Width = (endByteFromLine - startByteFromLine) * mMWidth
    textFirstLine.x = TextLeftSide+startByteFromLine*mMWidth //+firstLine.Width/2
    textFirstLine.y = startLine*mXHeight //+ firstLine.Height/2
end if

end if

fig = RectsToFigureShape(firstLine, middleLines, lastLine)
if fig <> nil then
    if HexIsFocus then
        fig.FillColor = HighlightColor
    else
        fig.fill = 0
        fig.Border = 100
        fig.BorderWidth = 2
        fig.BorderColor = hsv(HighlightColor.hue, HighlightColor.saturation, HighlightColor.value/1.2)
    end if
    g.DrawObject fig
end if

if VisibilityOfText = true then
    fig = RectsToFigureShape(textFirstLine, textMiddleLines, textLastLine)
    if fig <> nil then
        if TextIsFocus then
            fig.FillColor = HighlightColor
        else
            fig.fill = 0
            fig.Border = 100
            fig.BorderWidth = 2
            fig.BorderColor = hsv(HighlightColor.hue, HighlightColor.saturation, HighlightColor.value/1.2)
        end if
        g.DrawObject fig
    end if
end if

end if
End Sub

```



HexEditCanvas.PointInText:

Function PointInText(x as integer, y as integer) As boolean

    return mShowText and x > TextLeftSide

End Function

### HexEditCanvas.PointToByteOffset:

Function PointToByteOffset(x as integer, y as integer) As integer

    dim offset as Integer

    dim slotsIn as Integer

    dim bytesIn as Integer

    offset = (y \ mXHeight) \* mBytesPerLine

    if mShowText and mTextIsFocus then

        offset = offset + ((x - TextLeftSide) / mMWidth)

    else

        'offset = offset + (x - HexLeftSide) \ mMWidth

        slotsIn = (x - HexLeftSide) / mMWidth

        if slotsIn >= (HexAreaWidth \ mMWidth) then

            slotsIn = (HexAreaWidth \ mMWidth) - 2

        end if

        bytesIn = (slotsIn \ ((mBytesPerGroup \* 2) + 1)) \* mBytesPerGroup

        bytesIn = bytesIn + ((slotsIn mod ((mBytesPerGroup \* 2) + 1)) \ 2)

        offset = offset + bytesIn

    end if

    if offset + mFirstDisplayedOffset > GetTotalDataLength() then

        Return GetTotalDataLength()

    end if

    return offset + mFirstDisplayedOffset

End Function

### HexEditCanvas.RectsToFigureShape:

Protected Function RectsToFigureShape(ar1 as rectshape, ar2 as rectshape, ar3 as rectshape) As figureShape

    dim res as FigureShape

    if ar1 <> nil then

        res = new FigureShape

        res.AddLine ar1.x, ar1.y+ar1.Height, ar1.x, ar1.y

        res.AddLine ar1.x, ar1.y, ar1.x + ar1.Width, ar1.y

        res.AddLine ar1.x + ar1.Width, ar1.y, ar1.x + ar1.Width, ar1.y + ar1.Height

    if ar2 <> nil then

        res.AddLine ar1.x + ar1.Width, ar1.y + ar1.Height, ar1.x + ar1.Width, ar2.y + ar2.Height

    if ar3 <> nil and ar3.Height > 0 and ar3.Width > 0 then

        res.AddLine ar1.x + ar1.Width, ar2.y + ar2.Height, ar3.x+ar3.Width, ar2.y + ar2.Height

        res.AddLine ar3.x+ar3.Width, ar2.y + ar2.Height, ar3.x+ar3.Width, ar3.y + ar3.Height

        res.AddLine ar3.x+ar3.Width, ar3.y + ar3.Height, ar3.x, ar3.y + ar3.Height

        res.AddLine ar3.x, ar3.y + ar3.Height, ar3.x, ar2.y

        if ar3.x <> ar1.x then

            res.AddLine ar3.x, ar2.y, ar1.x, ar1.y + ar1.Height

```

        end if
    else
        res.AddLine ar1.x + ar1.Width, ar2.y + ar2.Height, ar2.x, ar2.y + ar2.Height
        res.AddLine ar2.x, ar2.y + ar2.Height, ar2.x, ar2.y
        if ar1.x <> ar2.x then
            res.AddLine ar2.x, ar2.y, ar1.x, ar1.y+ar1.Height
        end if
    end if
elseif ar3 <> nil and ar3.Height > 0 and ar3.Width > 0 then
    res.AddLine ar1.x + ar1.Width, ar1.y + ar1.Height, ar3.x+ar3.Width, ar3.y
    res.AddLine ar3.x+ar3.Width, ar3.y, ar3.x + ar3.Width, ar3.y + ar3.Height
    res.AddLine ar3.x + ar3.Width, ar3.y + ar3.Height, ar3.x, ar3.y + ar3.Height
    res.AddLine ar3.x, ar3.y + ar3.Height, ar3.x, ar3.y
    if ar3.x <> ar1.x then
        res.AddLine ar3.x, ar3.y, ar1.x, ar1.y + ar1.Height
    end if
else
    res.AddLine ar1.x + ar1.Width, ar1.y + ar1.Height, ar1.x, ar1.y + ar1.Height
end if
end if
Return res

```

End Function

### HexEditCanvas.RevealByte:

Sub RevealByte(aOffset as integer)

```

    if aOffset < mFirstDisplayedOffset then
        SetTopLine(aOffset\mBytesPerLine)
        me.Refresh
    elseif aOffset > (mFirstDisplayedOffset + (LinesDisplayed-1)*mBytesPerLine) then
        SetTopLine((aOffset - (LinesDisplayed-1)*mBytesPerLine)\mBytesPerLine)
        me.Refresh
    end if
End Sub

```

### HexEditCanvas.SanityChecks:

Protected Sub SanityChecks()

```

    dim td as Integer

    td = GetTotalDataLength()
    if mSelStart < 0 then
        mSelStart = 0
    end if
    if mSelStart > td then
        mSelStart = td
    end if
    if mSelStart + mSelLength > td then
        mSelLength = td - mSelStart
    end if

    if mFirstDisplayedOffset \ mBytesPerLine <> mFirstDisplayedOffset / mBytesPerLine then
        mFirstDisplayedOffset = (mFirstDisplayedOffset \ mBytesPerLine) * mBytesPerLine
    end if

```

End Sub

### **HexEditCanvas.SelectionLength:**

```
Function SelectionLength() As integer
    Return mSelLength
End Function
```

### **HexEditCanvas.SelectionStart:**

```
Function SelectionStart() As integer
    Return mSelStart
End Function
```

### **HexEditCanvas.SetDisplayCharacteristics:**

```
Protected Sub SetDisplayCharacteristics(aG as graphics)
    dim fname as String
    dim fsize as Integer
    dim oldSize as Integer

    oldSize = aG.TextSize

    fname = GetFont()
    if fname = "" then
        fname = kFontName
    end if
    aG.TextFont = fname

    fsize = GetFontSize()
    if fsize = 0 then
        fsize = kFontSize
    end if
    aG.TextSize = fsize

    if mLastFontName <> fname or oldSize <> fsize then
        CacheCommonData(aG)
        DetermineBadCharacters(fname, fsize, aG)
    end if
    mLastFontName = fname
End Sub
```

### **HexEditCanvas.SetDrawStripes:**

```
Sub SetDrawStripes(aDraw as boolean)
    mDrawStripes = aDraw
End Sub
```

### **HexEditCanvas.SetNumberOfCharactersForOffset:**

```
Sub SetNumberOfCharactersForOffset(aChars as integer)
    mCharactersForOffset = aChars
End Sub
```

### **HexEditCanvas.SetSelection:**

```
Sub SetSelection(aStart as integer, aLen as integer)
    dim oldStart, oldLen as Integer
```

```

oldStart = mSelStart
oldLen = mSelLength
if aLen < 0 then
    mSelLength = abs(aLen)
    mSelStart = aStart - mSelLength
else
    mSelStart = aStart
    mSelLength = aLen
end if
if mSelStart < 0 then
    mSelStart = 0
elseif mSelStart > GetTotalDataLength() then
    mSelStart = GetTotalDataLength()
end if

if mSelStart + mSelLength > GetTotalDataLength() then
    mSelLength = GetTotalDataLength() - mSelStart
end if
me.Refresh
End Sub

```

### **HexEditCanvas.SetTopLine:**

```

Sub SetTopLine(aLine as integer)
    dim line as Integer
    line = min(ceil(GetTotalDataLength / mBytesPerLine)-LinesDisplayed, aLine)
    line = max(0, line)

```

```

    mFirstDisplayedOffset = line * mBytesPerLine
End Sub

```

### **HexEditCanvas.SetVisibilityOfText:**

```

Sub SetVisibilityOfText(aVisibility as boolean)
    mShowText = aVisibility
End Sub

```

### **HexEditCanvas.SetWriteable:**

```

Sub SetWriteable(aWrite as boolean)
    mWriteable = aWrite
End Sub

```

### **HexEditCanvas.TextAreaWidth:**

```

Function TextAreaWidth() As integer
    Return mMWidth * mBytesPerLine
End Function

```

### **HexEditCanvas.TextIsFocus:**

```

Function TextIsFocus() As boolean
    Return mShowText and mTextIsFocus
End Function

```

### **HexEditCanvas.TextLeftSide:**

```

Function TextLeftSide() As integer

```

```
Return me.Width - (kRightMargin + mMWidth*mBytesPerLine)
End Function
```

### **HexEditCanvas.UpdateByteRange:**

```
Sub UpdateByteRange(aStart as integer, aLen as integer)
    me.Refresh
End Sub
```

### **HexEditCanvas.VisibilityOfText:**

```
Function VisibilityOfText() As boolean
    Return mShowText
End Function
Protected mBackColor As color
```

Protected mBadChars As string

Protected mBadCharsRE As regex

Protected mBytesPerGroup As integer

Protected mBytesPerLine As integer

Protected mCharactersForOffset As integer

Protected mClickStartOffset As integer

Protected mDataAsDisplayed As string

Protected mDividerColor As color

Protected mDrawStripes As boolean

Protected mFirstDisplayedOffset As integer

Protected mInsertionOn As boolean

Protected mLastFontName As string

Protected mLastInsertion As integer

Protected mMWidth As double

Protected mOffsetsAsDisplayed As string

Protected mOnLowNibble As boolean

Protected mSelDirection As integer

Protected mSelLength As integer

Protected mSelStart As integer

Protected mShowText As boolean

Protected mStripeColor As color

Protected mTextAsDisplayed As string

Protected mTextColor As color

Protected mTextIsFocus As boolean

Protected mWriteable As boolean

Protected mXHeight As double

End Class

## **Module HexEditCanvasStatics**

Const kACode = 65

Const kArrowDown = 31

Const kArrowLeft = 28

Const kArrowRight = 29

Const kArrowUp = 30

Const kDeleteKey = 8

Const kFCode = 70

Const kFontName = "Andale Mono"

Const kFontSize = 12

Const kInterTextSlop = 0

Const kLACode = 97

Const kLeftExtraSpace = 10

Const kLeftMargin = 5

Const kLFCode = 102

Const kNineCode = 57

Const kRightMargin = 15

Const kZeroCode = 48

### **HexEditCanvasStatics.HECConvertTextToBinary:**

Function HECConvertTextToBinary(aStr as string) As string

    'given hex as text (i.e. 7F1B) convert it to binary data

    dim newData as String

    dim re as RegEx

    dim rm as RegExMatch

    dim count as Integer

    dim sourceData, targetData as MemoryBlock

    dim n1, n2 as Integer

    if aStr = "" then

        Return ""

    end if

    re = new RegEx

    re.SearchPattern = "\s+"

    re.Options.ReplaceAllMatches = true

    re.Options.TreatTargetAsOneLine = true

    re.ReplacementPattern = ""

    newData = re.Replace(aStr)

```

if (lenb(newData) mod 2) <> 0 then
    Return ""
end if
re.SearchPattern = "[^0-9a-fA-F]"
rm = re.Search(newData)
if rm <> nil then
    Return ""
end if

newData = Uppercase(newData)
sourceData = NewMemoryBlock(lenb(newData))
targetData = NewMemoryBlock(lenb(newData)/2)
sourceData.StringValue(0, lenb(newData)) = newData
for count = 0 to sourceData.Size - 1 Step 2
    n1 = sourceData.Byte(count)
    n2 = sourceData.Byte(count+1)
    if n1 >= kACode then
        n1 = (n1 - kACode) + 10
    else
        n1 = n1 - kZeroCode
    end if
    if n2 >= kACode then
        n2 = (n2 - kACode) + 10
    else
        n2 = n2 - kZeroCode
    end if

    targetData.Byte(count/2) = (n1*16)+n2
next
return targetData.StringValue(0,targetData.Size)
End Function

```

### HexEditCanvasStatics.ReasonableFonts:

```

Sub ReasonableFonts(aFonts() as string)
    dim i as Integer
    dim fname as String
    dim p as Picture
    dim g as Graphics

    p = new Picture(16,16,32)
    g = p.Graphics
    Redim aFonts(-1)
    g.TextSize = 12
    for i = 0 to FontCount-1
        g.TextFont = Font(i)
        if g.StringWidth("iiiiiiii") = g.StringWidth("MMMMMMMMMM") then
            aFonts.Append Font(i)
        end if
    next
End Sub
End Module

```

### **bitmap\_class\_EX.Constructor:**

```
Sub Constructor(in_data as memoryblock, in_offset as integer, in_magic as integer)
    header = new bitm_header_EX
    magic = in_magic
    data_ref = in_data
    offset = in_offset
End Sub
```

### **bitmap\_class\_EX.read:**

```
Sub read()
    dim br as BinaryStream = new BinaryStream(data_ref)
    br.LittleEndian = true
    br.Position = 0
    header.read(br, magic, offset)

    br.Position = header.image_translation
    redim image_info(-1)
    for i as integer = 1 to header.image_count
        dim temp_image as new bitm_image_info
        temp_image.read(br)
        image_info.Append( temp_image )
    next
End Sub
```

### **bitmap\_class\_EX.update\_offset:**

```
Function update_offset(image_info_ind as integer, new_offset as integer, internal as boolean) As boolean
    if image_info_ind < 0 OR image_info_ind > UBound(image_info) then Return False

    image_info(image_info_ind).offset = new_offset
    if BitAnd(image_info(image_info_ind).flags, &h100) = &h100 then
        //external
        if internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    else
        //internal
        if not internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    end

    dim bw as BinaryStream = new BinaryStream(data_ref)
    bw.LittleEndian = true
    image_info(image_info_ind).write(bw)

    Return True
End Function
```

### **bitmap\_class\_EX.update\_offset:**



```

Function update_offset(byref bw as binarystream, image_info_ind as integer, new_offset as integer, internal as
boolean) As boolean
    if image_info_ind < 0 OR image_info_ind > UBound(image_info) then Return False

    image_info(image_info_ind).offset = new_offset
    if BitAnd(image_info(image_info_ind).flags, &h100) = &h100 then
        //external
        if internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    else
        //internal
        if not internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    end

    bw.LittleEndian = true
    image_info(image_info_ind).write(bw)

    Return True
End Function
data_ref As memoryBlock

header As bitm_header_EX

image_info(-1) As bitm_image_info

magic As Integer

offset As Integer

DDS resource

http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9\_c/directx/graphics/reference/
ddsfilereference/ddsfileformat.asp
End Class

```

## **Class bitmap\_class\_EX\_old**

### **bitmap\_class\_EX\_old.Constructor:**

```

Sub Constructor(in_bitmaps_f as folderItem, in_f as folderitem, in_offset as integer, in_magic as integer)
    bitmaps_f = in_bitmaps_f
    header = new bitm_header_RW
    magic = in_magic
    f = in_f
    offset = in_offset
End Sub

```

### **bitmap\_class\_EX\_old.extract\_DDS:**

```

Function extract_DDS(index as integer) As memoryBlock
    //most of this code was borrowed from the HMT source

```

```

dim dds as new MemoryBlock(0)

//Surface Format Header
dim dds_header as string = "DDS "

//DDSURFACEDESC2
dim size_of_structure as integer = 124
dim flags as integer = 0
dim height as integer
dim width as integer
dim PitchOrLinearSize as integer
dim depth as integer = image_info(index).depth
dim mipmapcount as integer = 0 'total number, not just the count
dim reserved1() as integer
dim reserved2 as integer = 0

flags = flags + int32(DDSEnum.DDSD_CAPS)
flags = flags + int32(DDSEnum.DDSD_PIXELFORMAT)
flags = flags + int32(DDSEnum.DDSD_WIDTH)
flags = flags + int32(DDSEnum.DDSD_HEIGHT)
select case image_info(index).format
case &hE, &hF, &h10
    flags = flags + int32(DDSEnum.DDSD_LINEARSIZE)
case else
    flags = flags + int32(DDSEnum.DDSD_PITCH)
end
if image_info(index).type = 1 then
    flags = flags + int32(DDSEnum.DDSD_DEPTH)
end
height = image_info(index).height
width = image_info(index).height

dim RGBBitCount as Integer = 0
select case image_info(index).format
case &h6 //R5G6B5
    RGBBitCount = 16
case &h8 //A1R4G5B5
    RGBBitCount = 16
case &h9 //A4R4G4B4
    RGBBitCount = 16
case &hA //X8R8G8B8
    RGBBitCount = 32
case &hB //A8R8G8B8
    RGBBitCount = 32
end select
select case image_info(index).format
case &hE, &hF, &h10 //linear
    //size in bytes of a single main image
    if image_info(index).format = int32(BitmFormat.DXT1) then
        PitchOrLinearSize = max(image_info(index).width/4, 1) * max(image_info(index).height/4, 1) * 8
    else
        PitchOrLinearSize = max(image_info(index).width/4, 1) * max(image_info(index).height/4, 1) * 16
    end
end

```

```

case else //pitch
    PitchOrLinearSize = image_info(index).width * (RGBBitCount/8)
end select

if image_info(index).num_mipmaps > 0 AND image_info(index).type <> 2 then
    mipmapcount = image_info(index).num_mipmaps + 1
    flags = flags + int32(DDSEnum.DDSD_MIPMAPCOUNT)
end

redim reserved1(-1)
for i as integer = 1 to 11
    reserved1.append(0)
next

//DDPIXELFORMAT
dim size as integer = 32
dim ddpixel_flags as integer = 0
dim FourCC as string
dim ddpixel_RGBBitCount as integer
dim RBitMask as integer
dim GBitMask as integer
dim BBitMask as integer
dim RGBAlphaBitMask as integer
select case image_info(index).format
case int32(BitmFormat.DXT1)
    FourCC = "DXT1"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case int32(BitmFormat.DXT2_3)
    FourCC = "DXT3"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case int32(BitmFormat.DXT4_5)
    FourCC = "DXT4"
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_FOURCC)
case else
    FourCC = chr(0) + chr(0) + chr(0) + chr(0)
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_RGB)
end select
select case image_info(index).format
case int32(BitmFormat.R5G6B5)
    ddpixel_RGBBitCount = 16
    RBitMask = &hF800
    GBitMask = &h7E0
    BBitMask = &h1F
    RGBAlphaBitMask = &h0
case int32(BitmFormat.A1R5G5B5)
    ddpixel_RGBBitCount = 16
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &h7C00
    GBitMask = &h3E0
    BBitMask = &h1F
    RGBAlphaBitMask = &h8000
case int32(BitmFormat.A4R4G4B4)
    ddpixel_RGBBitCount = 16
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)

```

```

RBitMask = &HF000
GBitMask = &HF0
BBitMask = &HF
RGBAlphaBitMask = &HF00000
case int32(BitmFormat.X8R8G8B8)
    ddpixel_RGBBitCount = 32
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &HFF000
    GBitMask = &HFF00
    BBitMask = &HFF
    RGBAlphaBitMask = &HFF000000
case int32(BitmFormat.A8R8G8B8)
    ddpixel_RGBBitCount = 32
    ddpixel_flags = ddpixel_flags + int32(DDSEnum.DDPF_ALPHAPIXELS)
    RBitMask = &HFF0000
    GBitMask = &HFF00
    BBitMask = &HFF
    RGBAlphaBitMask = &HFF000000
case else
    ddpixel_RGBBitCount = 0
    RBitMask = 0
    GBitMask = 0
    BBitMask = 0
    RGBAlphaBitMask = &H0
end select

//ddsCaps
dim caps1 as integer = 0
dim caps2 as integer = 0
dim reserved_DDSCAPS() as integer
caps1 = caps1 + int32(DDSEnum.DDSCAPS_TEXTURE)
if image_info(index).num_mipmaps > 0 then
    caps1 = caps1 + int32(DDSEnum.DDSCAPS_MIPMAP)
end
if image_info(index).num_mipmaps > 0 or image_info(index).type = 2 or image_info(index).type = 3 then
    caps1 = caps1 + int32(DDSEnum.DDSCAPS_COMPLEX)
end
if image_info(index).type = 2 then
    caps2 = caps2 + int32(DDSEnum.DDSCAPS2_CUBEMAP) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEX) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEX) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEY) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEY) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_POSITIVEZ) _
    + int32(DDSEnum.DDSCAPS2_CUBEMAP_NEGATIVEZ)
end
if image_info(index).type = 1 then
    caps2 = caps2 + int32(DDSEnum.DDSCAPS2_VOLUME)
end
redim reserved_DDSCAPS(-1)
reserved_DDSCAPS.Append(0)
reserved_DDSCAPS.Append(0)

//write the header data

```

```

dim bw as new BinaryStream(dds)
bw.LittleEndian = true
bw.Write(dds_header)
bw.WriteInt32(size_of_structure)
bw.WriteInt32(flags)
bw.WriteInt32(height)
bw.WriteInt32(width)
bw.WriteInt32(PitchOrLinearSize)
bw.WriteInt32(depth)
bw.WriteInt32(mipmapcount)
for i as integer = 0 to UBound(reserved1)
    bw.WriteInt32(reserved1(i))
next
//the pixel format and caps are structs contained in the surface so write them then reserved2
//pixel format
bw.WriteInt32(size)
bw.WriteInt32(ddpixel_flags)
bw.Write(FourCC)
bw.WriteInt32(ddpixel_RGBBitCount)
bw.WriteInt32(RBitMask)
bw.WriteInt32(GBitMask)
bw.WriteInt32(BBitMask)
bw.WriteInt32(AlphaBitMask)
//dds caps
bw.WriteInt32(caps1)
bw.WriteInt32(caps2)
for i as integer = 0 to UBound(reserved_DDSCAPS)
    bw.WriteInt32(reserved_DDSCAPS(i))
next
bw.WriteInt32(reserved2)

//now right the raw image data
dim br as BinaryStream
if BitAnd(image_info(index).flags, &h100) = &h100 then
    br = bitmaps_f.OpenAsBinaryFile
else
    br = f.OpenAsBinaryFile
end
br.LittleEndian = true
br.Position = image_info(index).offset
for i as integer = 1 to image_info(index).size step 4
    bw.WriteInt32(br.ReadInt32)
next

return dds
End Function

```

### **bitmap\_class\_EX\_old.gradient\_colors:**

```

Private Function gradient_colors(value1 as color, value2 as color) As color
    dim temp_color as color
    dim r as integer = ((value1.red*2) + value2.red)/3
    dim g as integer = ((value1.green*2) + value2.green)/3
    dim b as integer = ((value1.blue*2) + value2.blue)/3
    temp_color = rgb(r,g,b)

```

```
Return temp_color
End Function
```

### **bitmap\_class\_EX\_old.gradient\_colors\_half:**

```
Private Function gradient_colors_half(value1 as color, value2 as color) As color
    dim temp_color as color
    dim R as integer = (value1.Red/2) + (value2.Red/2)
    dim G as integer = (value1.Green/2) + (value2.Green/2)
    dim B as integer = (value1.Blue/2) + (value2.Blue/2)
    temp_color = RGB(R,G,B)
    Return temp_color
End Function
```

### **bitmap\_class\_EX\_old.inject\_DDS:**

```
Function inject_DDS(index as integer, in_data as memoryBlock, EOF as boolean = false, name as string = "Unknown")
As integer
```

```
    //use the integer return to specify whether it was sucessful or failed and why
```

```
    //0: Succeeded
    //-1: Failed, no known reason
    //-2: Failed, file size too large
    //-3: Failed, bad header
    //-4: Failed, image dimensions too large
    //-5: Failed, image wrong format
    //-6: Failed, needs to be a cubemap
    //-7: Failed, needs to be a volume texture
```

```
    //check to see if it's too large
    if in_data.Size - &h80 > image_info(index).size then
        return -2
    end
```

```
    dim br as new BinaryStream(in_data)
    br.LittleEndian = true
```

```
    //check the header
    br.Position = 0
    dim dds_header as string
    dds_header = br.read(4)
    if dds_header <> "DDS " then
        Return -3
    end
```

```
    //check the image size
    br.Position = 12
    dim width as int32 = br.ReadInt32
    dim height as int32 = br.ReadInt32
```

```
    if width > image_info(index).width or height > image_info(index).height then
        Return -4
    end
```

```
    //check the format
    br.Position = 80
```

```

dim ddpixel_flags as int32 = br.ReadInt32
dim FourCC as string = br.Read(4)
if BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_FOURCC)) = int32(DDSEnum.DDPF_FOURCC) then
    //dxt format
    select case FourCC
    case "DXT1"
        if image_info(index).format <> int32(BitmFormat.DXT1) then Return - 5

    case "DXT2", "DXT3"
        if image_info(index).format <> int32(BitmFormat.DXT2_3) then Return - 5

    case "DXT4", "DXT5"
        if image_info(index).format <> int32(BitmFormat.DXT4_5) then Return - 5

    end select
else
    //check to make sure the RGB format is correct
    br.Position = 88
    dim RGBbitcount as integer = br.ReadInt32
    dim Rbitmask as integer = br.ReadInt32
    dim Gbitmask as integer = br.ReadInt32
    dim Bbitmask as integer = br.ReadInt32
    dim RGBAlphaBitMask as integer = br.ReadInt32

    select case image_info(index).format
    case int32(BitmFormat.R5G6B5)
        if RGBbitcount <> 16 or _
            RBitMask <> &hF800 or _
            GBitMask <> &h7E0 or _
            BBitMask <> &h1F or _
            RGBAlphaBitMask <> &h0 then return -5
    case int32(BitmFormat.A1R5G5B5)
        if RGBbitcount <> 16 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _
            RBitMask <> &h7C00 or _
            GBitMask <> &h3E0 or _
            BBitMask <> &h1F or _
            RGBAlphaBitMask <> &h8000 then return -5
    case int32(BitmFormat.A4R4G4B4)
        if RGBbitcount <> 16 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _
            RBitMask <> &HF000 or _
            GBitMask <> &HF0 or _
            BBitMask <> &HF or _
            RGBAlphaBitMask <> &HF00000 then return -5
    case int32(BitmFormat.X8R8G8B8)
        //won't hold alpha channel against you
        if RGBbitcount <> 32 or _
            RBitMask <> &HFF000 or _
            GBitMask <> &HFF00 or _
            BBitMask <> &HFF then return -5
    case int32(BitmFormat.A8R8G8B8)
        if RGBbitcount <> 32 or _
            BitAnd(ddpixel_flags, int32(DDSEnum.DDPF_ALPHAPIXELS)) <> int32(DDSEnum.DDPF_ALPHAPIXELS) or _

```

```

        RBitMask <> &HFF0000 or _
        GBitMask <> &HFF00 or _
        BBitMask <> &HFF or _
        RGBAlphaBitMask <> &HFF000000 then return -5
    end select
end

//check for cubemappery
if image_info(index).type = 2 then
    br.position = 112
    dim ddscaps2 as int32 = br.readint32
    if BitAnd(ddscaps2, int32(DDSEnum.DDSCAPS2_CUBEMAP)) <> int32(DDSEnum.DDSCAPS2_CUBEMAP) then
        Return -6
    end
end
//volume checking
if image_info(index).type = 1 then
    br.position = 112
    dim ddscaps2 as int32 = br.readint32
    if BitAnd(ddscaps2, int32(DDSEnum.DDSCAPS2_VOLUME)) <> int32(DDSEnum.DDSCAPS2_VOLUME) then
        Return - 7
    end
    br.position = 8
    dim dds_flags as int32 = br.readint32
    if BitAnd(dds_flags, int32(DDSEnum.DDSD_DEPTH)) <> int32(DDSEnum.DDSD_HEIGHT) then
        Return -7
    end
end

//write the data
//maybe later add an option to inject into raw data?
if not EOF then
    dim bw as BinaryStream
    if BitAnd(image_info(index).flags, &h100) = &h100 then
        bw = bitmaps_f.OpenAsBinaryFile(true)
    else
        bw = f.OpenAsBinaryFile(true)
    end
    bw.LittleEndian = true
    bw.Position = image_info(index).offset
    br.Position = &h80
    for i as integer = 1 to in_data.Size - &h80 step 4
        bw.WriteInt32(br.ReadInt32)
    next
    bw.close
    br.close

    //update the info in the map
    bw = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = image_info(index).position + 4
    bw.Writeshort(width)
    bw.Writeshort(height)
    bw.Position = image_info(index).position + 28

```



```

    bw.WriteInt32(in_data.Size - &h80)
    bw.Close
end
if EOF then
    dim bitmap_map as new H1SupportMap.SupportMap
    bitmap_map.read(bitmaps_f)

    //ewww all this to truncate 80 bytes
    dim inject_data as new MemoryBlock(0)
    dim bw_data as new BinaryStream(inject_data)
    bw_data.LittleEndian = true
    dim br_data as new BinaryStream(in_data)
    br_data.LittleEndian = true
    br.Position = &h80
    for i as integer = 1 to in_data.Size - &h80 step 4
        bw_data.WriteInt32(br_data.ReadInt32)
    next
    bw_data.close
    br_data.close

    image_info(index).offset = bitmap_map.inject(inject_data, name)
    //make sure it's now part of bitmaps.map
    if BitAnd(image_info(index).flags, &h100) <> &h100 then
        image_info(index).flags = image_info(index).flags + &h100
    end
    //update the info in the map
    dim bw as BinaryStream
    bw = f.OpenAsBinaryFile(true)
    bw.LittleEndian = true
    bw.Position = image_info(index).position + 4
    bw.Writeshort(width)
    bw.Writeshort(height)
    bw.Position = image_info(index).position + 28
    bw.WriteInt32(in_data.Size - &h80)
    bw.Position = image_info(index).position + 14 //flags offset
    bw.WriteInt32(image_info(index).flags)
    bw.Position = image_info(index).position + 24 //write the updated offset
    bw.WriteInt32(offset)
    bw.Close
end

```

```

    return 0
End Function

```

### **bitmap\_class\_EX\_old.read:**

```

Sub read()
    dim br as BinaryStream = f.OpenAsBinaryFile
    br.LittleEndian = true
    br.Position = offset
    header.read(br, magic)

    br.Position = header.image_offset
    redim images(-1)

```

```

redim image_info(-1)
for i as integer = 1 to header.image_count
    dim temp_image as new bitm_image_info
    temp_image.read(br)
    image_info.Append( temp_image )
next
End Sub

```

### **bitmap\_class\_EX\_old.read\_2D:**

```

Private Sub read_2D(info as bitm_image_info)
    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_pic as Picture
        temp_pic = read_A8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_A8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_image.mip_maps.append(temp_mip_image)
        next
        images.Append temp_image

    case &h1 //Y8
        dim temp_pic as Picture
        temp_pic = read_Y8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_Y8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_image.mip_maps.append(temp_mip_image)
        next
        images.Append temp_image

    case &h2 //AY8
        dim temp_pixs() as Picture
        temp_pixs = read_AY8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
        for k as integer = 1 to info.num_mipmaps
            dim temp_mips() as Picture
            temp_mips() = read_AY8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
            temp_image.mip_maps.append(temp_mip_image)
        next
    end select
end Sub

```

```

images.Append temp_image

case &h3 //A8Y8
    dim temp_pixs() as Picture
    temp_pixs = read_A8Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h6 //R5G6B5
    dim temp_pic as Picture
    temp_pic = read_R5G6B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_R5G6B5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h8 //A1R5G5B5
    dim temp_pixs() as Picture
    temp_pixs = read_A1R5G5B5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h9 //A4R4G4B4
    dim temp_pixs() as Picture
    temp_pixs = read_A4R4G4B4(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hA //X8R8G8B8
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)

```

```

for k as integer = 1 to info.num_mipmaps
    dim temp_mip as Picture
    temp_mip = read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mip)
    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hB //A8R8G8B8
    dim temp_pixs() as Picture
    temp_pixs = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hE //DXT1
    dim temp_pic as Picture
    temp_pic = read_DXT1(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_DXT1(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hF //DXT2and3
    dim temp_pixs() as Picture
    temp_pixs = read_DXT2_3(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h10 //DXT4and5
    dim temp_pixs() as Picture
    temp_pixs = read_DXT4_5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))
        temp_image.mip_maps.append(temp_mip_image)
    next

```

```

images.Append temp_image

case &h11 //P8
    dim temp_pic as Picture
    temp_pic = read_P8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_P8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

end select
End Sub

```

### **bitmap\_class\_EX\_old.read\_3D:**

```

Private Sub read_3D(info as bitm_image_info)
    //so it's basically of the a similar format but unlike cubemaps, it reads each layer then mips
    //also, mips have half as many layers as the main, halving until there's only 1 layer per mip level

    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_pic as Picture
        temp_pic = read_A8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        dim layers(-1) as picture
        for i as integer = 2 to info.depth
            layers.append read_A8(info.width, info.height, br)
        next
        temp_image.add_layers(layers)

        for k as integer = 1 to info.num_mipmaps
            dim temp_mip as Picture
            temp_mip = read_A8(info.width/(2^k), info.height/(2^k), br)
            dim temp_mip_image as new bitm_image(temp_mip)

            dim mip_layers(-1) as Picture
            for j as integer = 2 to info.depth/(2^k)
                mip_layers.append read_A8(info.width/(2^k), info.height/(2^k), br)
            next
            temp_mip_image.add_layers(mip_layers)

            temp_image.mip_maps.append(temp_mip_image)
        next
    end select
End Sub

```

```

next
images.Append temp_image

case &h1 //Y8
dim temp_pic as Picture
temp_pic = read_Y8(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pic)
dim layers(-1) as picture
for i as integer = 2 to info.depth
    layers.append read_Y8(info.width, info.height, br)
next
temp_image.add_layers(layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mip as Picture
    temp_mip = read_Y8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mip)

    dim mip_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        mip_layers.append read_Y8(info.width/(2^k), info.height/(2^k), br)
    next
    temp_mip_image.add_layers(mip_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h2 //AY8
dim temp_pixs() as Picture
temp_pixs = read_AY8(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_AY8(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_AY8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_AY8(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

```

```

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h3 //A8Y8
dim temp_pixs() as Picture
temp_pixs = read_A8Y8(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A8Y8(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A8Y8(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h6 //R5G6B5
dim temp_pic as Picture
temp_pic = read_R5G6B5(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pic)
dim layers(-1) as picture
for i as integer = 2 to info.depth
    layers.append read_R5G6B5(info.width, info.height, br)
next
temp_image.add_layers(layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mip as Picture
    temp_mip = read_R5G6B5(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mip)

    dim mip_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        mip_layers.append read_R5G6B5(info.width/(2^k), info.height/(2^k), br)

```

```

    next
    temp_mip_image.add_layers(mip_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h8 //A1R5G5B5
dim temp_pixs() as Picture
temp_pixs = read_A1R5G5B5(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A1R5G5B5(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A1R5G5B5(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h9 //A4R4G4B4
dim temp_pixs() as Picture
temp_pixs = read_A4R4G4B4(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A4R4G4B4(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)

```



```

    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A4R4G4B4(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hA //X8R8G8B8
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_X8R8G8B8(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mip)

        dim mip_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            mip_layers.append read_X8R8G8B8(info.width/(2^k), info.height/(2^k), br)
        next
        temp_mip_image.add_layers(mip_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &hB //A8R8G8B8
    dim temp_pixs() as Picture
    temp_pixs = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_A8R8G8B8(info.width, info.height, br)
        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

    for k as integer = 1 to info.num_mipmaps

```

```

dim temp_mips() as Picture
temp_mips() = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

dim mip_layers(-1), mip_alpha_layers(-1) as Picture
for j as integer = 2 to info.depth/(2^k)
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_A8R8G8B8(info.width/(2^k), info.height/(2^k), br)
    mip_layers.append temp_layer_pixs(1)
    mip_alpha_layers.append temp_layer_pixs(0)
next
temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hE //DXT1
dim temp_pic as Picture
temp_pic = read_DXT1(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pic)
dim layers(-1) as picture
for i as integer = 2 to info.depth
    layers.append read_DXT1(info.width, info.height, br)
next
temp_image.add_layers(layers)

for k as integer = 1 to info.num_mipmaps
    dim temp_mip as Picture
    temp_mip = read_DXT1(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mip)

    dim mip_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        mip_layers.append read_DXT1(info.width/(2^k), info.height/(2^k), br)
    next
    temp_mip_image.add_layers(mip_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &hF //DXT2and3
dim temp_pixs() as Picture
temp_pixs = read_DXT2_3(info.width, info.height, br)
dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
dim layers(-1), alpha_layers(-1) as picture
for i as integer = 2 to info.depth
    dim temp_layer_pixs(-1) as picture
    temp_layer_pixs = read_DXT2_3(info.width, info.height, br)
    layers.append temp_layer_pixs(1)
    alpha_layers.append temp_layer_pixs(0)
next
temp_image.add_layers(layers, alpha_layers)

```

```

for k as integer = 1 to info.num_mipmaps
    dim temp_mips() as Picture
    temp_mips() = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
    dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

    dim mip_layers(-1), mip_alpha_layers(-1) as Picture
    for j as integer = 2 to info.depth/(2^k)
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_DXT2_3(info.width/(2^k), info.height/(2^k), br)
        mip_layers.append temp_layer_pixs(1)
        mip_alpha_layers.append temp_layer_pixs(0)
    next
    temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

    temp_image.mip_maps.append(temp_mip_image)
next
images.Append temp_image

case &h10 //DXT4and5
    dim temp_pixs() as Picture
    temp_pixs = read_DXT4_5(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pixs(1), temp_pixs(0))
    dim layers(-1), alpha_layers(-1) as picture
    for i as integer = 2 to info.depth
        dim temp_layer_pixs(-1) as picture
        temp_layer_pixs = read_DXT4_5(info.width, info.height, br)
        layers.append temp_layer_pixs(1)
        alpha_layers.append temp_layer_pixs(0)
    next
    temp_image.add_layers(layers, alpha_layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mips() as Picture
        temp_mips() = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitm_image(temp_mips(1), temp_mips(0))

        dim mip_layers(-1), mip_alpha_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            dim temp_layer_pixs(-1) as picture
            temp_layer_pixs = read_DXT4_5(info.width/(2^k), info.height/(2^k), br)
            mip_layers.append temp_layer_pixs(1)
            mip_alpha_layers.append temp_layer_pixs(0)
        next
        temp_mip_image.add_layers(mip_layers, mip_alpha_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

case &h11 //P8
    dim temp_pic as Picture
    temp_pic = read_P8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)

```

```

    dim layers(-1) as picture
    for i as integer = 2 to info.depth
        layers.append read_P8(info.width, info.height, br)
    next
    temp_image.add_layers(layers)

    for k as integer = 1 to info.num_mipmaps
        dim temp_mip as Picture
        temp_mip = read_P8(info.width/(2^k), info.height/(2^k), br)
        dim temp_mip_image as new bitmap_image(temp_mip)

        dim mip_layers(-1) as Picture
        for j as integer = 2 to info.depth/(2^k)
            mip_layers.append read_P8(info.width/(2^k), info.height/(2^k), br)
        next
        temp_mip_image.add_layers(mip_layers)

        temp_image.mip_maps.append(temp_mip_image)
    next
    images.Append temp_image

end select
End Sub

bitmap_class_EX_old.read_A1R5G5B5:
Private Function read_A1R5G5B5(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint16 = br.readuint16
            dim A as UInt16 = Bitwise.ShiftRight(value, 15) * &hFF
            dim R as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 10), &h1F) * &hFF)/31
            dim G as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 5), &h1F) * &hFF)/31
            dim B as UInt16 = (BitAnd(Bitwise.ShiftRight(value, 0), &h1F) * &hFF)/31
            p_color.RGBSurface.pixel(x,y) = RGB(r,g,b)
            p_alpha.RGBSurface.pixel(x,y) = RGB(a,a,a)
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

### **bitmap\_class\_EX\_old.read\_A4R4G4B4:**

```

Private Function read_A4R4G4B4(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

```

```

for y as integer = 0 to height-1
    for x as integer = 0 to width-1
        dim value as uint16 = br.readuint16
        dim A as integer = (Bitwise.ShiftRight(value, 12) * &hFF) / 15
        dim R as integer = (BitAnd(Bitwise.ShiftRight(value, 8), &h0F)*&hFF)/15
        dim G as integer = (BitAnd(Bitwise.ShiftRight(value, 4), &h0F)*&hFF)/15
        dim B as integer = (BitAnd(Bitwise.ShiftRight(value, 0), &h0F)*&hFF)/15
        dim temp_color as color = RGB(R, G, B)
        p_color.RGBSurface.pixel(X,Y) = temp_color
        p_alpha.RGBSurface.pixel(X,Y) = RGB(A,A,A)
    next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_EX\_old.read\_A8:**

```

Private Function read_A8(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            //value = Bitwise.ShiftLeft(value, 24)
            dim temp_color as color = RGB(value, value, value)
            p.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p
End Function

```

### **bitmap\_class\_EX\_old.read\_A8R8G8B8:**

```

Private Function read_A8R8G8B8(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint32 = br.readuint32
            dim A as integer = Bitwise.ShiftRight(value, 24, 32)
            dim R as integer = BitAnd(Bitwise.ShiftRight(value, 16, 32), 255)
            dim G as integer = BitAnd(Bitwise.ShiftRight(value, 8, 32), 255)
            dim B as integer = BitAnd(Bitwise.ShiftRight(value, 0, 32), 255)
            //this might just be a fancy way of reading each channel
            //as uint8
            dim temp_color as color = RGB(R, G, B)
            p_color.RGBSurface.pixel(X,Y) = temp_color
            p_alpha.RGBSurface.pixel(X,Y) = RGB(A,A,A)
        next
    next

```

```

    next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_EX\_old.read\_A8Y8:**

```

Private Function read_A8Y8(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim r_g_b as uint8 = Bitwise.ShiftRight(value, 8)
            dim alpha as uint8 = BitAnd(value, &hFF)
            p_color.RGBSurface.pixel(X,Y) = RGB(r_g_b, r_g_b, r_g_b)
            p_alpha.RGBSurface.pixel(X,Y) = RGB(alpha, alpha, alpha)
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

### **bitmap\_class\_EX\_old.read\_AY8:**

```

Private Function read_AY8(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim temp_color as color = RGB(value, value, value)
            p_color.RGBSurface.pixel(X,Y) = temp_color
            p_alpha.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    dim p(-1) as Picture
    p.append(p_alpha)
    p.append(p_color)
    return p
End Function

```

```

bitmap_class_EX_old.read_bitmap:
Sub read_bitmap(index as integer)
    redim images(-1)
    redim cube_images(-1)

    select case image_info(index).type
    case 0
        read_2D(image_info(index))
    case 1
        read_3D(image_info(index))
    case 2
        read_cube_maps(image_info(index))
    end select
End Sub

```

### **bitmap\_class\_EX\_old.read\_bitmaps:**

```

Sub read_bitmaps()
    redim images(-1)
    redim cube_images(-1)

    for i as integer = 0 to UBound(image_info)
        select case image_info(i).type
        case 0
            read_2D(image_info(i))
        case 1
            read_3D(image_info(i))
        case 2
            read_cube_maps(image_info(i))
        end select
    next
End Sub

```

### **bitmap\_class\_EX\_old.read\_cube\_maps:**

```

Private Sub read_cube_maps(info as bitm_image_info)
    dim br as BinaryStream
    if BitAnd(info.flags, &h100) = &h100 then
        br = bitmaps_f.OpenAsBinaryFile
    else
        br = f.OpenAsBinaryFile
    end
    br.LittleEndian = true
    br.Position = info.offset

    select case info.format
    case &h0 //A8
        dim temp_images() as bitm_image
        for i as integer = 0 to 5 //have to read all the faces first
            dim temp_pic as Picture
            temp_pic = read_A8(info.width, info.height, br)
            dim temp_image as new bitm_image(temp_pic)
            temp_images.append temp_image
        next
    end select
End Sub

```

```

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_A8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h1 //Y8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_Y8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_Y8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h2 //AY8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pix() as Picture
        temp_pix = read_AY8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_AY8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h3 //A8Y8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first

```



```

    dim temp_pix() as Picture
    temp_pix = read_A8Y8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A8Y8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h6 //R5G6B5
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_R5G6B5(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_R5G6B5(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h8 //A1R5G5B5
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pix() as Picture
        temp_pix = read_A1R5G5B5(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_A1R5G5B5(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next

```

```

dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h9 //A4R4G4B4
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_A4R4G4B4(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_A4R4G4B4(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hA //X8R8G8B8
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pic as Picture
    temp_pic = read_X8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_X8R8G8B8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hB //A8R8G8B8
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_A8R8G8B8(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5

```

```

        dim temp_mip() as Picture
        temp_mip = read_A8R8G8B8(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hE //DXT1
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pic as Picture
    temp_pic = read_DXT1(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pic)
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip as Picture
        temp_mip = read_DXT1(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip)
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &hF //DXT2and3
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_DXT2_3(info.width, info.height, br)
    dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
    temp_images.append temp_image
next

for j as integer = 1 to info.num_mipmaps
    for i as integer = 0 to 5
        dim temp_mip() as Picture
        temp_mip = read_DXT2_3(info.width/(2^j), info.height/(2^j), br)
        dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
        temp_images(i).mip_maps.append(temp_mip_image)
    next
next
dim temp_cube as new cube_map(temp_images)
cube_images.Append temp_cube

case &h10 //DXT4and5
dim temp_images() as bitm_image
for i as integer = 0 to 5 //have to read all the faces first
    dim temp_pix() as Picture
    temp_pix = read_DXT4_5(info.width, info.height, br)

```

```

        dim temp_image as new bitm_image(temp_pix(1), temp_pix(0))
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip() as Picture
            temp_mip = read_DXT4_5(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip(1), temp_mip(0))
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

case &h11 //P8
    dim temp_images() as bitm_image
    for i as integer = 0 to 5 //have to read all the faces first
        dim temp_pic as Picture
        temp_pic = read_P8(info.width, info.height, br)
        dim temp_image as new bitm_image(temp_pic)
        temp_images.append temp_image
    next

    for j as integer = 1 to info.num_mipmaps
        for i as integer = 0 to 5
            dim temp_mip as Picture
            temp_mip = read_P8(info.width/(2^j), info.height/(2^j), br)
            dim temp_mip_image as new bitm_image(temp_mip)
            temp_images(i).mip_maps.append(temp_mip_image)
        next
    next
    dim temp_cube as new cube_map(temp_images)
    cube_images.Append temp_cube

end select
End Sub

```

### **bitmap\_class\_EX\_old.read\_DXT1:**

```

Private Function read_DXT1(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = new Picture(width, height, 32)

    dim color1,color2,color3,color4 as color
    dim zero_color as Color = RGB(0,0,0)
    dim chunkspersHline as integer = Width/4
    if chunkspersHline = 0 then chunkspersHline = 1

    dim i as integer = 0

    while i < width*height
        dim c1, c2 as UInt16
        dim trans as boolean

```

```

c1 = br.ReadUInt16
c2 = br.ReadUInt16
if c1 > c2 then
    trans = false
else
    trans = true
end
color1 = uint16_to_color(c1)
color2 = uint16_to_color(c2)
if not trans then
    color3 = gradient_colors(color1, color2)
    color4 = gradient_colors(color2, color1)
else
    color3 = gradient_colors_half(color1, color2)
    color4 = zero_color
end

dim cdata as UInt64 = br.ReadUInt32
dim ChunkNum as UInt32 = i / 16
dim XPos as UInt32 = ChunkNum mod chunksperHline
dim YPos as UInt32 = (ChunkNum - XPos) / chunksperHline

dim sizeW, sizeH as integer

if Height < 4 then
    sizeH = Height
else
    sizeH = 4
end
if Width < 4 then
    sizeW = Width
else
    sizeW = 4
end

for y as integer = 0 to sizeH-1
    for x as integer = 0 to sizeW-1
        dim temp_color as color
        select case BitAnd(cdata, &h3)
            case 0
                temp_color = color1
            case 1
                temp_color = color2
            case 2
                temp_color = color3
            case 3
                temp_color = color4
            case else
                break
        end select
        p.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = temp_color
        cdata = Bitwise.ShiftRight(cdata, 2)
    next
next

```

```
    i = i + 16
wend
```

```
Return p
End Function
```

### **bitmap\_class\_EX\_old.read\_DXT2\_3:**

```
Private Function read_DXT2_3(width as integer, height as integer, br as binaryStream) As picture()
```

```
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)
```

```
    dim chunkspersHline as integer = width/4
    if chunkspersHline = 0 then chunkspersHline = 1
```

```
    for i as integer = 0 to (width*height)-1 step 16
        dim Alpha(-1) as UInt16
        for j as integer = 0 to 3
            Alpha.append br.ReadUInt16
        next
        dim colors(-1) as color
        dim c1 as uint16 = br.readuint16
        dim c2 as uint16 = br.readuint16
        if c2 >= c1 then
            //dim temp as integer <- for breakpoints
        end
        colors.Append uint16_to_color(c1)
        colors.Append uint16_to_color(c2)
        colors.Append gradient_colors(colors(0), colors(1))
        colors.Append gradient_colors(colors(1), colors(0))
        dim value as UInt32 = br.ReadUInt32
        dim ChunkNum as UInt32 = i/16
        dim XPos as UInt32 = ChunkNum mod chunkspersHline
        dim YPos as UInt32 = (ChunkNum - XPos)/chunkspersHline
        dim sizeh, sizew as integer
        if height < 4 then sizeh = height else sizeh = 4
        if width < 4 then sizew = width else sizew = 4
```

```
        for y as integer = 0 to sizeh - 1
            for x as integer = 0 to sizew - 1
                dim temp_color as color = colors(BitAnd(value, &h03))
                value = Bitwise.ShiftRight(value, 2)
                dim alpha_value as integer = BitAnd(Alpha(y), &h0F) * 17
                dim alpha_color as color = RGB(alpha_value, alpha_value, alpha_value)
                Alpha(y) = Bitwise.ShiftRight(Alpha(y), 4)
                p_color.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = temp_color
                p_alpha.RGBSurface.pixel(x+(XPos*4), y+(YPos*4)) = alpha_color
            next
        next
    next
```

```
    dim p(-1) as Picture
```

```

p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_EX\_old.read\_DXT4\_5:**

```

Private Function read_DXT4_5(width as integer, height as integer, br as binaryStream) As picture()
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    dim p_alpha as Picture = NewPicture(width, height, 32)

    dim chunkspersHline as integer = width/4
    if chunkspersHline = 0 then chunkspersHline = 1

    for i as integer = 0 to (width*height) - 1 step 16
        dim alpha(7) as UInt8
        alpha(0) = br.ReadUInt8
        alpha(1) = br.ReadUInt8
        dim temp_word as UInt16 = br.ReadUInt16
        dim temp_dword as UInt32 = br.ReadUInt32
        dim AlphaDat as UInt32 = BitOr(temp_word, Bitwise.ShiftRight(temp_dword, 16)) //might need to alter this due
        to endianness
        if alpha(0) > alpha(1) then
            //8-alpha block: derive the other size alphas
            //Bit code 000 = alpha_0, 001 = alpha_1, others are interpolated.
            Alpha(2) = (6 * Alpha(0) + 1 * Alpha(1) + 3) / 7 // bit code 010
            Alpha(3) = (5 * Alpha(0) + 2 * Alpha(1) + 3) / 7 // bit code 011
            Alpha(4) = (4 * Alpha(0) + 3 * Alpha(1) + 3) / 7 // bit code 100
            Alpha(5) = (3 * Alpha(0) + 4 * Alpha(1) + 3) / 7 // bit code 101
            Alpha(6) = (2 * Alpha(0) + 5 * Alpha(1) + 3) / 7 // bit code 110
            Alpha(7) = (1 * Alpha(0) + 6 * Alpha(1) + 3) / 7 // bit code 111
        else
            //6-alpha block.
            //Bit code 000 = alpha_0, 001 = alpha_1, others are interpolated.
            Alpha(2) = (4 * Alpha(0) + 1 * Alpha(1) + 2) / 5 // Bit code 010
            Alpha(3) = (3 * Alpha(0) + 2 * Alpha(1) + 2) / 5 // Bit code 011
            Alpha(4) = (2 * Alpha(0) + 3 * Alpha(1) + 2) / 5 // Bit code 100
            Alpha(5) = (1 * Alpha(0) + 4 * Alpha(1) + 2) / 5 // Bit code 101
            Alpha(6) = 0 // Bit code 110
            Alpha(7) = 255 // Bit code 111
        end
        dim colors(-1) as color
        colors.Append(uint16_to_color(br.ReadUInt16))
        colors.Append(uint16_to_color(br.ReadUInt16))
        colors.Append(gradient_colors(colors(0), colors(1)))
        colors.Append(gradient_colors(colors(1), colors(0)))
        dim value as UInt32 = br.ReadUInt32

        dim ChunkNum as UInt32 = i/16
        dim XPos as UInt32 = ChunkNum mod chunkspersHline
        dim YPos as UInt32 = (ChunkNum - XPos)/chunkspersHline
        dim sizeh, sizew as integer
        if height < 4 then sizeh = height else sizeh = 4
    
```

```

if width < 4 then sizew = width else sizew = 4

for x as integer = 0 to sizeh -1
    for y as integer = 0 to sizew - 1
        dim temp_color as color = colors(BitAnd(value, &h03))
        value = Bitwise.ShiftRight(value, 2)
        dim alpha_value as UInt8 = alpha(BitAnd(AlphaDat, &h07))
        AlphaDat = Bitwise.ShiftRight(AlphaDat, 3)
        p_color.RGBSurface.pixel(y+(XPos*4), x+(YPos*4)) = temp_color
        p_alpha.RGBSurface.pixel(y+(XPos*4), x+(YPos*4)) = RGB(alpha_value, alpha_value, alpha_value)
    next
next
next

dim p(-1) as Picture
p.append(p_alpha)
p.append(p_color)
return p
End Function

```

### **bitmap\_class\_EX\_old.read\_P8:**

```

Private Function read_P8(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint8 = br.readuint8
            dim temp_color as color = RGB(value, value, value)
            p.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p
End Function

```

### **bitmap\_class\_EX\_old.read\_R5G6B5:**

```

Private Function read_R5G6B5(width as integer, height as integer, br as binaryStream) As picture
    if width = 0 then width = 1
    if height = 0 then height = 1
    dim p_color as Picture = NewPicture(width, height, 32)
    for y as integer = 0 to height-1
        for x as integer = 0 to width-1
            dim value as uint16 = br.readuint16
            dim temp_color as color = uint16_to_color(value)
            p_color.RGBSurface.pixel(X,Y) = temp_color
        next
    next

    return p_color
End Function

```



**bitmap\_class\_EX\_old.read\_X8R8G8B8:**

Private Function read\_X8R8G8B8(width as integer, height as integer, br as binaryStream) As picture

if width = 0 then width = 1

if height = 0 then height = 1

dim p\_color as Picture = NewPicture(width, height, 32)

for y as integer = 0 to height-1

for x as integer = 0 to width-1

dim value as uint32 = br.readuint32

dim R as integer = BitAnd(Bitwise.ShiftRight(value, 16, 32), 255)

dim G as integer = BitAnd(Bitwise.ShiftRight(value, 8, 32), 255)

dim B as integer = BitAnd(Bitwise.ShiftRight(value, 0, 32), 255)

//this might just be a fancy way of reading each channel

//as uint8

dim temp\_color as color = RGB(R, G, B)

p\_color.RGBSurface.pixel(X,Y) = temp\_color

next

next

return p\_color

End Function

**bitmap\_class\_EX\_old.read\_Y8:**

Private Function read\_Y8(width as integer, height as integer, br as binaryStream) As picture

if width = 0 then width = 1

if height = 0 then height = 1

dim p as Picture = NewPicture(width, height, 32)

for y as integer = 0 to height-1

for x as integer = 0 to width-1

dim value as uint8 = br.readuint8

dim temp\_color as color = RGB(value, value, value)

p.RGBSurface.pixel(X,Y) = temp\_color

next

next

return p

End Function

**bitmap\_class\_EX\_old.uint16\_to\_color:**

Private Function uint16\_to\_color(input as uint16) As color

dim temp\_color as color

dim R as integer = (BitAnd(Bitwise.ShiftRight(input, 11), &h1F) \* &hFF)/31

dim G as integer = (BitAnd(Bitwise.ShiftRight(input, 5), &h3F) \* &hFF)/63

dim B as integer = (BitAnd(Bitwise.ShiftRight(input, 0), &h1F) \* &hFF)/31

temp\_color = RGB(R,G,B)

return temp\_color

End Function

**bitmap\_class\_EX\_old.update\_offset:**

Function update\_offset(image\_info\_ind as integer, new\_offset as integer, internal as boolean) As boolean

if image\_info\_ind < 0 OR image\_info\_ind > UBound(image\_info) then Return False

image\_info(image\_info\_ind).offset = new\_offset

if BitAnd(image\_info(image\_info\_ind).flags, &h100) = &h100 then

```

    //external
    if internal then
        image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
    end
else
    //internal
    if not internal then
        image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
    end
end

dim bw as BinaryStream = f.OpenAsBinaryFile(true)
bw.LittleEndian = true
image_info(image_info_ind).write(bw)

Return True
End Function

bitmap_class_EX_old.update_offset:
Function update_offset(byref bw as binarystream, image_info_ind as integer, new_offset as integer, internal as
boolean) As boolean
    if image_info_ind < 0 OR image_info_ind > UBound(image_info) then Return False

    image_info(image_info_ind).offset = new_offset
    if BitAnd(image_info(image_info_ind).flags, &h100) = &h100 then
        //external
        if internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    else
        //internal
        if not internal then
            image_info(image_info_ind).flags = BitXor(image_info(image_info_ind).flags, &h100)
        end
    end

    bw.LittleEndian = true
    image_info(image_info_ind).write(bw)

    Return True
End Function
bitmaps_f As folderItem

cube_images(-1) As cube_map

f As folderItem

header As bitm_header_RW

images(-1) As bitm_image

image_info(-1) As bitm_image_info

magic As Integer

```

offset As Integer

DDS resource

[http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9\\_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp](http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/ddsfilereference/ddsfileformat.asp)

End Class

## **Class bitm\_header EX**

### **bitm\_header\_EX.read:**

Sub read(byref br as binaryStream, magic as integer, offset as integer)

    br.LittleEndian = true

    redim unknown(-1)

    for i as integer = 1 to 22

        unknown.append br.ReadInt32

    next

    translation\_to\_first = br.ReadInt32 - magic - offset

    unknown23 = br.ReadInt32

    image\_count = br.ReadInt32

    image\_translation = br.ReadInt32 - magic - offset

    unknown25 = br.ReadInt32

End Sub

image\_count As Integer

image\_translation As Integer

translation\_to\_first As Integer

unknown() As Integer

unknown23 As Integer

unknown25 As Integer

End Class

## **Class Progress\_Window**

Inherits Window

### **Progress\_Window.Constructor:**

Sub Constructor(title\_string as string)

    // Calling the overridden superclass constructor.

    Super.Window

    me.Title = title\_string

End Sub

### **Progress\_Window.tick:**

Sub tick(status as string, value as integer)

    ProgressBar1.Value = value

```

    StaticText2.text = status + ": " + str(value) + "%"
    //self.Refresh
End Sub
End Class

```

## **Class main\_control\_RWS**

Inherits ContainerControl

### **main\_control\_RWS.Close:**

```

Sub Close()
    if container_ref <> nil then
        container_ref.kill
    end
End Sub

```

### **main\_control\_RWS.Constructor:**

```

Sub Constructor(f as folderItem, in_offset as integer, in_magic as integer)
    fs = f
    offset = in_offset
    magic = in_magic //for passing along to reflexive editor controls
End Sub

```

### **main\_control\_RWS.Destructor:**

```

Sub Destructor()
    while UBound(bitmask16s) > -1
        bitmask16s(0).close
        bitmask16s.Remove(0)
    wend
    while UBound(bitmask32s) > -1
        bitmask32s(0).close
        bitmask32s.Remove(0)
    wend
    while UBound(bitmask16s) > -1
        bitmask8s(0).close
        bitmask8s.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorARGBs(0).Close
        colorARGBs.Remove(0)
    wend
    while UBound(colorbytes) > -1
        colorbytes(0).Close
        colorbytes.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorRGBs(0).Close
        colorRGBs.Remove(0)
    wend
    while UBound(dependencys) > -1
        dependencys(0).close
        dependencys.Remove(0)
    wend

```

```

while UBound(doubles) > -1
    doubles(0).close
    doubles.Remove(0)
wend
while UBound(floats) > -1
    floats(0).close
    floats.Remove(0)
wend
while UBound(id16s) > -1
    id16s(0).close
    id16s.Remove(0)
wend
while UBound(id32s) > -1
    id32s(0).close
    id32s.Remove(0)
wend
while UBound(id8s) > -1
    id8s(0).close
    id8s.Remove(0)
wend
while UBound(indexs) > -1
    indexs(0).Close
    indexs.Remove(0)
wend
while UBound(int16s) > -1
    int16s(0).close
    int16s.Remove(0)
wend
while UBound(int32s) > -1
    int32s(0).close
    int32s.Remove(0)
wend
while UBound(int8s) > -1
    int8s(0).close
    int8s.Remove(0)
wend
while UBound(loneIDs) > -1
    loneIDs(0).Close
    loneIDs.Remove(0)
wend
while UBound(reflexives) > -1
    reflexives(0).close
    reflexives.Remove(0)
wend
while UBound(string128s) > -1
    string128s(0).close
    string128s.Remove(0)
wend
while UBound(string32s) > -1
    string32s(0).close
    string32s.Remove(0)
wend
while UBound(string4s) > -1
    string4s(0).close

```

```

        String4s.Remove(0)
    wend
End Sub

```

### **main\_control\_RWS.init:**

```

Sub init(f_xml as folderItem, in_entity_style as boolean, in_show_hidden as boolean, in_edit_by_offset as boolean,
byref in_map as H1.map)
    entity_style = in_entity_style
    show_hidden = in_show_hidden
    edit_by_offset = in_edit_by_offset
    map = in_map

    init_left = me.left

    load(f_xml)
End Sub

```

### **main\_control\_RWS.load:**

```

Sub load(f_xml as folderItem)
    if container_ref <> nil then
        container_ref.kill
        container_ref = nil
    end
    while UBound(bitmask16s) > -1
        bitmask16s(0).close
        bitmask16s.Remove(0)
    wend
    while UBound(bitmask32s) > -1
        bitmask32s(0).close
        bitmask32s.Remove(0)
    wend
    while UBound(bitmask16s) > -1
        bitmask8s(0).close
        bitmask8s.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorARGBs(0).Close
        colorARGBs.Remove(0)
    wend
    while UBound(colorbytes) > -1
        colorbytes(0).Close
        colorbytes.Remove(0)
    wend
    while UBound(colorARGBs) > -1
        colorRGBs(0).Close
        colorRGBs.Remove(0)
    wend
    while UBound(dependencys) > -1
        dependencys(0).close
        dependencys.Remove(0)
    wend
    while UBound(doubles) > -1
        doubles(0).close
        doubles.Remove(0)
    wend

```

```

wend
while UBound(floats) > -1
    floats(0).close
    floats.Remove(0)
wend
while UBound(id16s) > -1
    id16s(0).close
    id16s.Remove(0)
wend
while UBound(id32s) > -1
    id32s(0).close
    id32s.Remove(0)
wend
while UBound(id8s) > -1
    id8s(0).close
    id8s.Remove(0)
wend
while UBound(indexs) > -1
    indexs(0).Close
    indexs.Remove(0)
wend
while UBound(int16s) > -1
    int16s(0).close
    int16s.Remove(0)
wend
while UBound(int32s) > -1
    int32s(0).close
    int32s.Remove(0)
wend
while UBound(int8s) > -1
    int8s(0).close
    int8s.Remove(0)
wend
while UBound(loneIDs) > -1
    loneIDs(0).Close
    loneIDs.Remove(0)
wend
while UBound(reflexives) > -1
    reflexives(0).close
    reflexives.Remove(0)
wend
while UBound(string128s) > -1
    string128s(0).close
    string128s.Remove(0)
wend
while UBound(string32s) > -1
    string32s(0).close
    string32s.Remove(0)
wend
while UBound(string4s) > -1
    string4s(0).close
    String4s.Remove(0)
wend

```

```

redim bitmask16s(-1)
redim bitmask32s(-1)
redim bitmask8s(-1)
redim colorARGBs(-1)
redim colorbytes(-1)
redim colorRGBs(-1)
redim dependencys(-1)
redim doubles(-1)
redim floats(-1)
redim id16s(-1)
redim id32s(-1)
redim id8s(-1)
redim indexs(-1)
redim int16s(-1)
redim int32s(-1)
redim int8s(-1)
redim loneIDs(-1)
redim reflexives(-1)
redim string128s(-1)
redim string32s(-1)
redim string4s(-1)

change_ok = false
dim f_top as FolderItem = GetFolderItem("").Child("Plugins")
dim class_str as string = f_xml.name.mid(1,4)
PopupMenu1.DeleteAllRows
for i as integer = 1 to f_top.Count
    dim f_xml_folder as FolderItem = f_top.item(i)
    if f_xml_folder.exists and f_xml_folder.Directory then
        if f_xml_folder.Child(class_str + ".ent").Exists then
            dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".ent")
            PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
            PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
        end
        if f_xml_folder.Child(class_str + ".xml").Exists then
            dim f_xml_temp as FolderItem = f_xml_folder.Child(class_str + ".xml")
            PopupMenu1.AddRow(f_xml_folder.Name + ":" + f_xml_temp.Name)
            PopupMenu1.RowTag(PopupMenu1.ListCount - 1) = f_xml_folder.Name + ":" + f_xml_temp.Name
        end
    end
end
next
for i as integer = 0 to PopupMenu1.ListCount - 1
    if PopupMenu1.RowTag(i) = f_xml.Parent.Name + ":" + f_xml.Name then
        PopupMenu1.ListIndex = i
        exit for i
    end
end
next
change_ok = true

dim plugin_info as new PluginLibUniversal
dim ref(-1) as integer
if entity_style then
    plugin_info.refresh_as_Ent(f_xml, false, ref, show_hidden)
else

```



```

    plugin_info.refresh_as_HMT(f_xml, false, ref)
end

//so organize the various editing components of the main reflexive
dim index_list(-1) as string
dim offset_list(-1) as integer
dim order_list(-1) as integer
for i as integer = 0 to UBound(plugin_info.bitmask16_offset)
    index_list.Append("bitmask16:" + str(i))
    offset_list.Append plugin_info.bitmask16_offset(i)
    order_list.Append plugin_info.bitmask16_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask32_offset)
    index_list.Append("bitmask32:" + str(i))
    offset_list.Append plugin_info.bitmask32_offset(i)
    order_list.Append plugin_info.bitmask32_order(i)
next
for i as integer = 0 to UBound(plugin_info.bitmask8_offset)
    index_list.Append("bitmask8:" + str(i))
    offset_list.Append plugin_info.bitmask8_offset(i)
    order_list.Append plugin_info.bitmask8_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorARGB_offset)
    index_list.Append("colorARGB:" + str(i))
    offset_list.Append plugin_info.colorARGB_offset(i)
    order_list.Append plugin_info.colorARGB_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorbyte_offset)
    index_list.Append("colorbyte:" + str(i))
    offset_list.Append plugin_info.colorbyte_offset(i)
    order_list.Append plugin_info.colorbyte_order(i)
next
for i as integer = 0 to UBound(plugin_info.colorRGB_offset)
    index_list.Append("colorRGB:" + str(i))
    offset_list.Append plugin_info.colorRGB_offset(i)
    order_list.Append plugin_info.colorRGB_order(i)
next
for i as integer = 0 to UBound(plugin_info.dependency_offset)
    index_list.Append("dependency:" + str(i))
    offset_list.Append plugin_info.dependency_offset(i)
    order_list.Append plugin_info.dependency_order(i)
next
for i as integer = 0 to UBound(plugin_info.double_offset)
    index_list.Append("double:" + str(i))
    offset_list.Append plugin_info.double_offset(i)
    order_list.Append plugin_info.double_order(i)
next
for i as integer = 0 to UBound(plugin_info.float_offset)
    index_list.Append("float:" + str(i))
    offset_list.Append plugin_info.float_offset(i)
    order_list.Append plugin_info.float_order(i)
next
for i as integer = 0 to UBound(plugin_info.id16_offset)
    index_list.Append("id16:" + str(i))

```

```

        offset_list.Append plugin_info.id16_offset(i)
        order_list.Append plugin_info.id16_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.id32_offset)
        index_list.Append("id32:" + str(i))
        offset_list.Append plugin_info.id32_offset(i)
        order_list.Append plugin_info.id32_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.id8_offset)
        index_list.Append("id8:" + str(i))
        offset_list.Append plugin_info.id8_offset(i)
        order_list.Append plugin_info.id8_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.index_offset)
        index_list.Append("index:" + str(i))
        offset_list.Append plugin_info.index_offset(i)
        order_list.Append plugin_info.index_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.int16_offset)
        index_list.Append("int16:" + str(i))
        offset_list.Append plugin_info.int16_offset(i)
        order_list.Append plugin_info.int16_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.int32_offset)
        index_list.Append("int32:" + str(i))
        offset_list.Append plugin_info.int32_offset(i)
        order_list.Append plugin_info.int32_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.int8_offset)
        index_list.Append("int8:" + str(i))
        offset_list.Append plugin_info.int8_offset(i)
        order_list.Append plugin_info.int8_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.loneID_offset)
        index_list.Append("loneID:" + str(i))
        offset_list.Append plugin_info.loneID_offset(i)
        order_list.Append plugin_info.loneID_order(i)
    next
    for i as Integer = 0 to UBound(plugin_info.reflexive_offset)
        index_list.Append("reflexive:" + str(i))
        offset_list.Append plugin_info.reflexive_offset(i)
        order_list.Append plugin_info.reflexive_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.string128_offset)
        index_list.Append("string128:" + str(i))
        offset_list.Append plugin_info.string128_offset(i)
        order_list.Append plugin_info.string128_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.string32_offset)
        index_list.Append("string32:" + str(i))
        offset_list.Append plugin_info.string32_offset(i)
        order_list.Append plugin_info.string32_order(i)
    next
    for i as integer = 0 to UBound(plugin_info.string4_offset)

```

```

index_list.Append("string4:" + str(i))
offset_list.Append plugin_info.string4_offset(i)
order_list.Append plugin_info.string4_order(i)
next

if edit_by_offset then
    offset_list.sortwith(index_list)
else
    order_list.sortwith(index_list, offset_list)
end

//now create the individual controls and add them to the main window control
container_ref = new Container_Scroller
container_ref.Width = me.Width - 5
container_ref.Height = me.Height - 49
//dim temp as integer = me.left
//break
container_ref.EmbedWithin(me, me.left - init_left, 49)
//dim temp_height as integer = 14
for i as integer = 0 to UBound(offset_list)
    dim temp_split_str() as string = split(index_list(i), ":")
    select case temp_split_str(0)

case "bitmask16"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask16_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask16_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_bitmask as new bitmask16_editor_control_RWS(fs, offset_list(i),
    plugin_info.bitmask16_data(val(temp_split_str(1))).labels, temp_ints)
    temp_bitmask.name_text.Text = plugin_info.bitmask16_name(val(temp_split_str(1)))
    //temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
    bitmask16s.Append temp_bitmask
    //temp_height = temp_height + 12 + temp_bitmask.Height
    container_ref.add(temp_bitmask)

case "bitmask32"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask32_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask32_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_bitmask as new bitmask32_editor_control_RWS(fs, offset_list(i),
    plugin_info.bitmask32_data(val(temp_split_str(1))).labels, temp_ints)
    temp_bitmask.name_text.Text = plugin_info.bitmask32_name(val(temp_split_str(1)))
    //temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
    bitmask32s.Append temp_bitmask
    //temp_height = temp_height + 12 + temp_bitmask.Height
    container_ref.add(temp_bitmask)

case "bitmask8"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.bitmask8_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.bitmask8_data(val(temp_split_str(1))).values(j) )
    next

```

```

dim temp_bitmask as new bitmask8_editor_control_RWS(fs, offset_list(i),
plugin_info.bitmask8_data(val(temp_split_str(1))).labels, temp_ints)
temp_bitmask.name_text.Text = plugin_info.bitmask8_name(val(temp_split_str(1)))
//temp_bitmask.EmbedWithin(Canvas1, 20, temp_height)
bitmask8s.Append temp_bitmask
//temp_height = temp_height + 12 + temp_bitmask.Height
container_ref.add(temp_bitmask)

case "colorARGB"
dim temp_color as new colorARGB_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorARGB_name(val(temp_split_str(1)))
//temp_color.EmbedWithin(Canvas1, 20, temp_height)
colorARGBs.Append temp_color
//temp_height = temp_height + 12 + temp_color.Height
container_ref.add(temp_color)

case "colorbyte"
dim temp_color as new colorbyte_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorbyte_name(val(temp_split_str(1)))
//temp_color.EmbedWithin(Canvas1, 20, temp_height)
colorbytes.Append temp_color
//temp_height = temp_height + 12 + temp_color.Height
container_ref.add(temp_color)

case "colorRGB"
dim temp_color as new colorRGB_editor_control_RWS(fs, offset_list(i))
temp_color.name_text.Text = plugin_info.colorRGB_name(val(temp_split_str(1)))
//temp_color.EmbedWithin(Canvas1, 20, temp_height)
colorRGBs.Append temp_color
//temp_height = temp_height + 12 + temp_color.Height
container_ref.add(temp_color)

case "dependency"
dim temp_dep as new dependency_editor_control_RWS(fs, offset_list(i), map)
temp_dep.name_text.Text = plugin_info.dependency_name(val(temp_split_str(1)))
//temp_dep.EmbedWithin(Canvas1, 20, temp_height)
dependencys.Append temp_dep
//temp_height = temp_height + 12 + temp_dep.Height
container_ref.add(temp_dep)

case "double"
dim temp_double as new double_editor_control_RWS(fs, offset_list(i))
temp_double.name_text.Text = plugin_info.double_name(val(temp_split_str(1)))
//temp_double.EmbedWithin(Canvas1, 20, temp_height)
doubles.Append temp_double
//temp_height = temp_height + 12 + temp_double.Height
container_ref.add(temp_double)

case "float"
dim temp_float as new float_editor_control_RWS(fs, offset_list(i))
temp_float.name_text.Text = plugin_info.float_name(val(temp_split_str(1)))
//temp_float.EmbedWithin(Canvas1, 20, temp_height)
floats.Append temp_float
//temp_height = temp_height + 12 + temp_float.Height

```

```

    container_ref.add(temp_float)

case "id16"
    dim temp_ints(-1) as Int16
    for j as integer = 0 to UBound(plugin_info.id16_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id16_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id16_editor_control_RWS(fs, offset_list(i),
    plugin_info.id16_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id16_name(val(temp_split_str(1)))
    //temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id16s.Append temp_enum
    //temp_height = temp_height + 12 + temp_enum.Height
    container_ref.add(temp_enum)

case "id32"
    dim temp_ints(-1) as Integer
    for j as integer = 0 to UBound(plugin_info.id32_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id32_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id32_editor_control_RWS(fs, offset_list(i),
    plugin_info.id32_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id32_name(val(temp_split_str(1)))
    //temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id32s.Append temp_enum
    //temp_height = temp_height + 12 + temp_enum.Height
    container_ref.add(temp_enum)

case "id8"
    dim temp_ints(-1) as Int8
    for j as integer = 0 to UBound(plugin_info.id8_data(val(temp_split_str(1))).values)
        temp_ints.Append( plugin_info.id8_data(val(temp_split_str(1))).values(j) )
    next
    dim temp_enum as new id8_editor_control_RWS(fs, offset_list(i),
    plugin_info.id8_data(val(temp_split_str(1))).labels, temp_ints)
    temp_enum.name_text.Text = plugin_info.id8_name(val(temp_split_str(1)))
    //temp_enum.EmbedWithin(Canvas1, 20, temp_height)
    id8s.Append temp_enum
    //temp_height = temp_height + 12 + temp_enum.Height
    container_ref.add(temp_enum)

case "index"
    dim temp_index as new index_editor_control_RWS(fs, offset_list(i),
    plugin_info.index_data(val(temp_split_str(1))))
    temp_index.name_text.Text = plugin_info.index_name(val(temp_split_str(1)))
    //temp_index.EmbedWithin(Canvas1, 20, temp_height)
    indexes.Append temp_index
    //temp_height = temp_height + 12 + temp_index.Height
    container_ref.add(temp_index)

case "int16"
    dim temp_int as new int16_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int16_name(val(temp_split_str(1)))
    //temp_int.EmbedWithin(Canvas1, 20, temp_height)

```

```

int16s.Append temp_int
//temp_height = temp_height + 12 + temp_int.Height
container_ref.add(temp_int)

case "int32"
    dim temp_int as new int32_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int32_name(val(temp_split_str(1)))
    //temp_int.EmbedWithin(Canvas1, 20, temp_height)
    int32s.Append temp_int
    //temp_height = temp_height + 12 + temp_int.Height
    container_ref.add(temp_int)

case "int8"
    dim temp_int as new int8_editor_control_RWS(fs, offset_list(i))
    temp_int.name_text.Text = plugin_info.int8_name(val(temp_split_str(1)))
    //temp_int.EmbedWithin(Canvas1, 20, temp_height)
    int8s.Append temp_int
    //temp_height = temp_height + 12 + temp_int.Height
    container_ref.add(temp_int)

case "lonelD"
    dim temp_dep as new lonelD_editor_control_RWS(fs, offset_list(i), map)
    temp_dep.name_text.Text = plugin_info.lonelD_name(val(temp_split_str(1)))
    //temp_dep.EmbedWithin(Canvas1, 20, temp_height)
    LonelDs.Append temp_dep
    //temp_height = temp_height + 12 + temp_dep.Height
    container_ref.add(temp_dep)

case "reflexive"
    dim temp_reflexive as new reflexive_control_RWS(fs, offset_list(i),
    plugin_info.reflexives(val(temp_split_str(1))).this_reflexive_size, magic, _
    plugin_info.reflexive_name_offset(val(temp_split_str(1))), offset)
    temp_reflexive.groupbox1.caption = plugin_info.reflexive_name(val(temp_split_str(1)))
    //temp_reflexive.EmbedWithin(Canvas1, 20, temp_Height)
    container_ref.add(temp_reflexive)
    dim height_diff as integer = temp_reflexive.height
    temp_reflexive.add_granddaddy(me)
    temp_reflexive.init(plugin_info.reflexives(val(temp_split_str(1))), edit_by_offset, map)
    height_diff = temp_reflexive.height - height_diff
    container_ref.increase_dim(height_diff)
    reflexives.Append temp_reflexive
    //temp_height = temp_height + 12 + temp_reflexive.Height

case "string128"
    dim temp_string as new string128_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string128_name(val(temp_split_str(1)))
    //temp_string.EmbedWithin(Canvas1, 20, temp_height)
    string128s.Append temp_string
    //temp_height = temp_height + 12 + temp_string.Height
    container_ref.add(temp_string)

case "string32"
    dim temp_string as new string32_editor_control_RWS(fs, offset_list(i))
    temp_string.name_text.Text = plugin_info.string32_name(val(temp_split_str(1)))

```

```

        //temp_string.EmbedWithin(Canvas1, 20, temp_height)
        string32s.Append temp_string
        //temp_height = temp_height + 12 + temp_string.Height
        container_ref.add(temp_string)

    case "string 4"
        dim temp_string as new string4_editor_control_RWS(fs, offset_list(i))
        temp_string.name_text.Text = plugin_info.string4_name(val(temp_split_str(1)))
        //temp_string.EmbedWithin(Canvas1, 20, temp_height)
        string4s.Append temp_string
        //temp_height = temp_height + 12 + temp_string.Height
        container_ref.add(temp_string)

    end select
next

//so now all of the stuff has been added to the control
//temp_height = temp_height + 8 // -12 from the last control +20 space it from the bottom
//me.Height = temp_height + 49
'dim temp_width as integer = 400 //the default width of editing controls
'for i as integer = 0 to UBound(reflexives)
'if reflexives(i).Width > temp_width then
'temp_width = reflexives(i).Width
'end
'next
'temp_width = temp_width + 10
'Canvas1.Refresh
'Canvas1.Width = temp_width
'me.Width = temp_width

read
//update_indexs
'if window isa Main_Window then
'Main_Window(window).SmartSplitter1.attachNearbyControls
'end
End Sub

```

### **main\_control\_RWS.read:**

```

Sub read()
    update_reflexive_offset
    update_indexs
    update_reflexive_offset
End Sub

```

### **main\_control\_RWS.update\_indexs:**

```

Sub update_indexs()
    //this only works for top level indexs
    for i as integer = 0 to UBound(indexs)
        dim temp_path() as string = split(indexs(i).path, ".")
        dim temp_reflexives() as reflexive_control_RWS = reflexives
        for j as integer = 1 to UBound(temp_path) - 1
            dim ref_name as string = temp_path(j)
            for k as integer = 0 to UBound(temp_reflexives)
                if temp_reflexives(k).groupbox1.caption = ref_name then

```

```

        temp_reflexives = temp_reflexives(k).reflexives
    exit for k
end
next
next
for j as integer = 0 to UBound(temp_reflexives)
    if temp_reflexives(j).groupbox1.caption = temp_path(UBound(temp_path)) then
        indexs(i).reflexive_pointed_at = temp_reflexives(j)
    exit for j
end
next
next
next
for i as integer = 0 to UBound(reflexives)
    reflexives(i).update_indexs
next
End Sub

```

### **main\_control\_RWS.update\_reflexive\_offset:**

```

Sub update_reflexive_offset()
    //their own "offset" is the local offset, the reflexive_offset is where the reflexive starts
    for i as integer = 0 to UBound(bitmask16s)
        bitmask16s(i).reflexive_offset = offset
        bitmask16s(i).read
        bitmask16s(i).Enabled = true
    next
    for i as integer = 0 to UBound(bitmask32s)
        bitmask32s(i).reflexive_offset = offset
        bitmask32s(i).read
        bitmask32s(i).Enabled = true
    next
    for i as integer = 0 to UBound(bitmask8s)
        bitmask8s(i).reflexive_offset = offset
        bitmask8s(i).read
        bitmask8s(i).Enabled = true
    next
    for i as integer = 0 to UBound(colorARGBs)
        colorARGBs(i).reflexive_offset = offset
        colorARGBs(i).read
        colorARGBs(i).Enabled = true
    next
    for i as integer = 0 to UBound(colorbytes)
        colorbytes(i).reflexive_offset = offset
        colorbytes(i).read
        colorbytes(i).Enabled = true
    next
    for i as integer = 0 to UBound(colorRGBs)
        colorRGBs(i).reflexive_offset = offset
        colorRGBs(i).read
        colorRGBs(i).Enabled = true
    next
    for i as integer = 0 to UBound(dependencys)
        dependencys(i).reflexive_offset = offset
        dependencys(i).read
    next
end Sub

```



```

    dependencys(i).Enabled = true
next
for i as integer = 0 to UBound(doubles)
    doubles(i).reflexive_offset = offset
    doubles(i).read
    doubles(i).Enabled = true
next
for i as integer = 0 to UBound(floats)
    floats(i).reflexive_offset = offset
    floats(i).read
    floats(i).Enabled = true
next
for i as integer = 0 to UBound(id16s)
    id16s(i).reflexive_offset = offset
    id16s(i).read
    id16s(i).Enabled = true
next
for i as integer = 0 to UBound(id32s)
    id32s(i).reflexive_offset = offset
    id32s(i).read
    id32s(i).Enabled = true
next
for i as integer = 0 to UBound(id8s)
    id8s(i).reflexive_offset = offset
    id8s(i).read
    id8s(i).Enabled = true
next
//would normally put indexs here but they need to be after reflexives
for i as integer = 0 to UBound(int16s)
    int16s(i).reflexive_offset = offset
    int16s(i).read
    int16s(i).Enabled = true
next
for i as integer = 0 to UBound(int32s)
    int32s(i).reflexive_offset = offset
    int32s(i).read
    int32s(i).Enabled = true
next
for i as integer = 0 to UBound(int8s)
    int8s(i).reflexive_offset = offset
    int8s(i).read
    int8s(i).Enabled = true
next
for i as integer = 0 to UBound(LoneIDs)
    loneIDs(i).reflexive_offset = offset
    loneIDs(i).read
    loneIDs(i).Enabled = true
next
for i as integer = 0 to UBound(reflexives)
    reflexives(i).reflexive_offset = offset
    reflexives(i).read
    reflexives(i).Enabled = true
next
for i as integer = 0 to UBound(string128s)

```

```

        string128s(i).reflexive_offset = offset
        string128s(i).read
        string128s(i).Enabled = true
    next
    for i as integer = 0 to UBound(string32s)
        string32s(i).reflexive_offset = offset
        string32s(i).read
        string32s(i).Enabled = true
    next
    for i as integer = 0 to UBound(string4s)
        string4s(i).reflexive_offset = offset
        string4s(i).read
        string4s(i).Enabled = true
    next
    for i as integer = 0 to UBound(indexs)
        indexs(i).reflexive_offset = offset
        indexs(i).read
        indexs(i).Enabled = true
    next
End Sub
bitmask16s() As bitmask16_editor_control_RWS

bitmask32s() As bitmask32_editor_control_RWS

bitmask8s() As bitmask8_editor_control_RWS

change_ok As boolean

colorARGBs() As colorARGB_editor_control_RWS

colorbytes() As colorbyte_editor_control_RWS

colorRGBs() As colorRGB_editor_control_RWS

container_ref As container_Scroller

dependencys() As dependency_editor_control_RWS

doubles() As double_editor_control_RWS

edit_by_offset As boolean

entity_style As boolean

floats() As float_editor_control_RWS

fs As folderItem

id16s() As id16_editor_control_RWS

id32s() As id32_editor_control_RWS

id8s() As id8_editor_control_RWS

```

indexs() As index\_editor\_control\_RWS

index\_data As string

### **main\_control\_RWS.init\_left:**

init\_left As Integer = 255

//255 is the original value found

int16s() As int16\_editor\_control\_RWS

int32s() As int32\_editor\_control\_RWS

int8s() As int8\_editor\_control\_RWS

lonelDs() As lonelD\_editor\_control\_RWS

magic As Integer

map As h1.map

offset As Integer

reflexives() As reflexive\_control\_RWS

show\_hidden As boolean

string128s() As string128\_editor\_control\_RWS

string32s() As string32\_editor\_control\_RWS

string4s() As string4\_editor\_control\_RWS

### **main\_control\_RWS Control PopupMenu1:**

Sub Change()

if not change\_ok then return

dim temp\_strs() as string = split(me.RowTag(me.ListIndex), ":")

if temp\_strs.ubound <> 1 then return

dim folder as FolderItem = GetFolderItem("").Child("Plugins").Child(temp\_strs(0))

if Folder.Exists and Folder.Directory then

dim plugin as FolderItem = Folder.Child(temp\_strs(1))

if plugin.Exists then

dim temp\_split() as string = plugin.name.split(".")

dim temp\_str as string

if temp\_split.Ubound < 1 then

'dim temp\_split2() as string = plugin.AbsolutePath.Split(":")

'dim temp\_str2 as string = temp\_split2(temp\_split2.Ubound)

'dim temp\_split3() as string = temp\_str2.split(".")

'if temp\_split3.Ubound < 1 then

'break

'errorbox("There was an error determining plugin extension type")

'return

'end

'dim temp\_str3 as string = temp\_split3(1)

```

        'endif buildtargetwindows
        'temp_str3 = temp_str3.left(temp_str3.length - 1)
        'endif
        'temp_str = temp_str3
        break
        return
    else
        temp_str = temp_split(1)
    end
    select case temp_str.Uppercase
    case "ENT"
        entity_style = true
    case "XML"
        entity_style = false
    end select
    load(plugin)
end
end
End Sub
End Class

```

## **Class Waiting\_Window**

Inherits Window

### **Waiting\_Window.Constructor:**

```

Sub Constructor(title_string as string)
    // Calling the overridden superclass constructor.
    Super.Window
    me.Title = title_string
End Sub

```

### **Waiting\_Window.tick:**

```

Sub tick(status as string)
    StaticText2.text = status
    //self.Refresh
End Sub
End Class

```

## **Class Tag\_Delete\_Window**

Inherits Window

### **Tag\_Delete\_Window.Constructor:**

```

Sub Constructor(in_main as Main_Window, in_map_index as integer, in_tag_index as integer)
    // Calling the overridden superclass constructor.
    Super.Window
    main = in_main
    map_index = in_map_index
    tag_index = in_tag_index
End Sub
main As main_window

```

map\_index As Integer

tag\_index As Integer

### **Tag\_Delete\_Window Control PushButton1:**

Sub Action()

```
dim success_bool as boolean = main.maps_expanded(map_index).remove_tag(tag_index)
if success_bool then
    main.ListBox1.ListIndex = 0
    dim temp_bool as boolean = main.tag_selected(0,0)
    main.update_list
end
self.close
```

End Sub

### **Tag\_Delete\_Window Control PushButton2:**

Sub Action()

```
self.close
```

End Sub

End Class

## **Class SBSP\_class\_EX**

### **SBSP\_class\_EX.Constructor:**

Sub Constructor(in\_data as memoryBlock, in\_magic as integer)

```
data_ref = in_data
magic = in_magic
start = new SBSP_start
header = new SBSP_header
redim dobc_refs(-1)
```

End Sub

### **SBSP\_class\_EX.read:**

Sub read()

```
dim br as new BinaryStream(data_ref)
br.LittleEndian = true
```

```
//read the beginning section
br.Position = 0
start = new SBSP_start
start.read(br, magic)
```

```
//read the header
br.Position = start.header_offset
header = new SBSP_header
header.read(br, magic)
```

```
redim dobc_refs(-1)
```

```

    br.Position = header.DOBC_reflexive.translation
    for i as integer = 0 to header.DOBC_reflexive.count - 1
        dim temp_dobc as new SBSP_DOBC_data
        temp_dobc.read(br, magic)
        dobc_refs.Append temp_dobc
    next
End Sub
data_ref As memoryBlock

dobc_refs(-1) As SBSP_DOBC_data

header As SBSP_header

magic As Integer

start As SBSP_start

End Class

```

## **Class SBSP\_start**

### **SBSP\_start.read:**

```

Sub read(br as binaryStream, magic as integer)
    header_offset = br.ReadUInt32 - magic
    xbox_vert_reflexive_count = br.ReadUInt32
    xbox_vert_reflexive_start = br.ReadUInt32 //unknown, always zero in PC maps
    xbox_lightmap_vert_reflexive_count = br.ReadUInt32
    xbox_lightmap_vert_reflexive_start = br.ReadUInt32 //likewise unknown and zero in PC
    sbsp_str = br.Read(4)
End Sub
header_offset As Integer

sbsp_str As string

xbox_lightmap_vert_reflexive_count As Integer

xbox_lightmap_vert_reflexive_start As Integer

xbox_vert_reflexive_count As Integer

xbox_vert_reflexive_start As Integer

End Class

```

## **Class SBSP\_header**

### **SBSP\_header.Constructor:**

```

Sub Constructor()
    DOBC_reflexive = new H1.reflexive
End Sub

```

```

SBSP_header.read:
Sub read(br as binaryStream, magic as integer)
    position = br.Position

    //skip ahead for now

    //Detail Object Collection references
    br.Position = position + &h24C
    DOBC_reflexive = new H1.reflexive
    DOBC_reflexive.offset = br.Position
    DOBC_reflexive.count = br.ReadUInt32
    DOBC_reflexive.translation = br.ReadUInt32 - magic

    //Decal references
    br.Position = position + &h258
    decal_reflexive = new H1.reflexive
    decal_reflexive.offset = br.Position
    decal_reflexive.count = br.ReadUInt32
    decal_reflexive.translation = br.ReadUInt32 - magic

    //End of the header
    br.Position = position + &h288
End Sub
decal_reflexive As H1.reflexive

DOBC_reflexive As H1.reflexive

position As Integer

End Class

```

## **Class SBSP\_DNBC\_data**

### **SBSP\_DNBC\_data.Constructor:**

```

Sub Constructor()
    unknown_reflexive_1 = new H1.reflexive
    unknown_reflexive_2 = new H1.reflexive
    unknown_reflexive_3 = new H1.reflexive
    unknown_reflexive_4 = new H1.reflexive

End Sub

```

### **SBSP\_DNBC\_data.read:**

```

Sub read(br as binaryStream, magic as integer)
    position = br.Position

    br.Position = position + &h0
    unknown_reflexive_1 = new H1.reflexive
    unknown_reflexive_1.offset = br.Position
    unknown_reflexive_1.count = br.ReadUInt32
    unknown_reflexive_1.translation = br.ReadUInt32 - magic

```

```

br.Position = position + &hC
unknown_reflexive_2 = new H1.reflexive
unknown_reflexive_2.offset = br.Position
unknown_reflexive_2.count = br.ReadUInt32
unknown_reflexive_2.translation = br.ReadUInt32 - magic

br.Position = position + &h18
unknown_reflexive_3 = new H1.reflexive
unknown_reflexive_3.offset = br.Position
unknown_reflexive_3.count = br.ReadUInt32
unknown_reflexive_3.translation = br.ReadUInt32 - magic

br.Position = position + &h24
unknown_reflexive_4 = new H1.reflexive
unknown_reflexive_4.offset = br.Position
unknown_reflexive_4.count = br.ReadUInt32
unknown_reflexive_4.translation = br.ReadUInt32 - magic

br.Position = position + &h64//the total chunk size
End Sub
position As Integer

unknown_reflexive_1 As h1.reflexive

unknown_reflexive_2 As h1.reflexive

unknown_reflexive_3 As h1.reflexive

unknown_reflexive_4 As h1.reflexive

End Class

```

## **Module RBGL**

```

Private Const AGL_LIB = ""
Private Const CARBON_LIB = ""
Private Const GLU_LIB = ""
Private Const GL_LIB = ""
Private Const RADIANS_TO_DEGREES = 57.2957795
Protected Const Version = 1.62
Private Const WINGDI_LIB = ""
Private Const WINUSER_LIB = ""

```

### **RBGL.private\_has\_mask:**

```

Private Function private_has_mask(image as Picture) As boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    #if RBVersion < 2008.02 then
        if image.Depth < 32 then

```



```

        return false
    else
        dim opaque as Boolean = true
        dim s as RGBSurface = image.mask.RGBSurface
        for i as uint32 = image.Height - 1 downto 0
            for j as uint32 = image.Width - 1 downto 0
                if s.pixel(j,i) <> &c000000 then
                    opaque = false
                    exit
                end
            next
        next
        return not opaque
    end
#else
    return image.Mask(false) <> nil
#endif
End Function

```

### **RBGL.private\_round\_to\_power\_of\_two:**

Private Function private\_round\_to\_power\_of\_two(length as uint32) As uint32

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //round length up to the next highest power of two
    const log2 = 0.693147180559945
    dim l as uint32 = log(length) / log2
    if pow(2,l) < length then
        return pow(2,l+1)
    else
        return length
    end
End Function

```

### **RBGL.RGBA:**

Function RGBA(red As UInt8, green As UInt8, blue As UInt8, alpha As UInt8) As RBGLColor

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    dim col as RBGLColor
    col.Red = red
    col.Green = green
    col.Blue = blue
    col.Alpha = alpha
    return col
End Function

```

### **RBGL.RGBA:**

Function RGBA(rgba As UInt32) As RBGLColor

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//convert a single 32-bit integer to a color
//allows hex input of colours as &hRRGGBBAA

static colm as new MemoryBlock(4)
colm.LittleEndian = false
colm.UInt32Value(0) = rgba

dim col as RBGLColor
col.StringValue(colm.LittleEndian) = colm

return col
End Function

```

### **RBGL.RGBA:**

```

Function RGBA(rgb As Color, alpha As UInt8 = 255) As RBGLColor
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//build an RGBA color from an RGB color and (optional) alpha value

static colm as new MemoryBlock(4)
colm.LittleEndian = false
colm.ColorValue(0,24) = rgb
colm.UInt8Value(3) = alpha

dim col as RBGLColor
col.StringValue(colm.LittleEndian) = colm

return col
End Function

```

### **RBGL.RGBColor:**

```

Function RGBColor(Extends col As RBGLColor) As Color
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

return RGB(col.Red,col.Green,col.Blue)
End Function
Private Delegate Function wglChoosePixelFormatARBDelegate(hdc as UInt32, piAttribIList as Ptr, piAttribFList as Ptr, nMaxFormats as UInt32, byref piFormats as UInt32, byref nNumFormats as UInt32) As Integer
Private Delegate Function wglCreatePbufferARBDelegate(hdc as UInt32, iPixelFormat as uint32, iWidth as uint32, iHeight as uint32, piAttributeList as Ptr) As UInt32
Private Delegate Function wglGetExtensionsStringARBDelegate(hdc as UInt32) As CString
Private Delegate Function wglGetPbufferDCARBDelegate(pbuffer as UInt32) As UInt32
Private private_current_context As WeakRef

```

Private private\_current\_graphics As WeakRef

Private private\_deleted\_textures\_by\_context As Dictionary

Global Enum RBGLErrorCode As UInt32

NoError = 0

InvalidEnum = &h0500

InvalidValue = &h0501

InvalidOperation = &h0502

StackOverflow = &h0503

StackUnderflow = &h0504

OutOfMemory = &h0505

TableTooLarge = &h8031

End Enum

Global Enum RBGLFeature As Integer

DepthTesting = &h0B71

StencilTesting = &h0B90

End Enum

Global Enum RBGLPixelFormat As UInt32

RGBA = 6408

RGB = 6407

Alpha = 6406

Luminance = 6409

LuminanceAlpha = 6410

End Enum

Global Enum RBGLScalingMode As UInt32

Linear = &h2601

Nearest = &h2600

End Enum

Global Enum RBGLWrapMode As UInt32

Clamp = 10496

Repeat = 10497

End Enum

Private Structure GLIntRect

Left as UInt32

Top as UInt32

Width as UInt32

Height as UInt32

End Structure

Global Structure RBGLColor

Red as UInt8

Green as UInt8

Blue as UInt8

Alpha as UInt8

End Structure

Global Structure RBGLContextOptions

Oversampling As UInt8

DepthBufferBits As UInt8

StencilBufferBits As UInt8

AccumulationBufferBits As UInt8

End Structure

Global Structure RBGLPictureOptions

ScaleToPowerOfTwo As Boolean

```

GenerateMipMaps As Boolean
WrapX As RBGLWrapMode
WrapY As RBGLWrapMode
Upscaling As RBGLScalingMode
Downscaling As RBGLScalingMode
End Structure
Class RBGLCanvas
Inherits Canvas

```

```

    Event Error(Error as RBGLErrorCode)
    Sub ()
    Event Init(ByRef Options As RBGLContextOptions)
    Sub ()
    Event Open()
    Sub ()
    Event Paint(g As RBGLGraphics)
    Sub ()

```

### **RBGLCanvas.Open:**

```

Sub Open()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //erase background
    EraseBackgroundGL = EraseBackground

    //prevent flicker on Windows
    EraseBackground = false

    //call init event to get custom options
    dim options as RBGLContextOptions = RBGLContext.DefaultOptions
    Init options

    //create context
    private_context = new RBGLContext(me,options)

    //call open event
    Open
End Sub

```

### **RBGLCanvas.Paint:**

```

Sub Paint(g As Graphics)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    Refresh EraseBackgroundGL
End Sub

```

### **RBGLCanvas.GetPicture:**

```

Function GetPicture(X As Integer, Y As Integer, Width As Integer, Height As Integer) As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get realbasic picture
    return Graphics.GetPicture(X, Y, Width, Height)

```

End Function

### **RBGLCanvas.GetPicture:**

```

Function GetPicture() As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get realbasic picture
    return Graphics.GetPicture(0, 0, -10000, -10000)

```

End Function

### **RBGLCanvas.GetRBGLPicture:**

```

Function GetRBGLPicture(X As Integer = 0, Y As Integer = 0, Width As Integer = -10000, Height As Integer = -10000) As RBGLPicture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get rbglpicture
    return Graphics.GetRBGLPicture(X, Y, Width, Height)

```

End Function

### **RBGLCanvas.private\_draw\_backdrop:**

```

Protected Sub private_draw_backdrop(g as RBGLGraphics, backdrop as Picture)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //overload this method if backdrop needs to be drawn in a special way, e.g. for sprite surface

    if StretchBackdrop then
        g.DrawPicture Backdrop, 0, 0, Width, Height, 0, 0, backdrop.Width, backdrop.Height
    else
        g.DrawPicture Backdrop, 0, 0
    end
End Sub

```

### **RBGLCanvas.Refresh:**

```

Sub Refresh(EraseBackground As Boolean = True)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //hide if invisible
    dim height as integer = me.Height
    if not Visible then
        me.height = 0
    end

    //bind context and get graphics object
    dim g as RBGLGraphics = Graphics

    //erase background
    if EraseBackground then private_context.Clear

    //draw backdrop texture
    if Backdrop <> nil then
        private_draw_backdrop g, Backdrop
    end

    //call paint event
    Paint g

    //detect errors
    do
        dim err as RBGLErrorCode = private_context.LastError
        if err = RBGLErrorCode.NoError then
            exit
        else
            Error err
        end
    loop

    //flush buffer (swap front and back buffers on Windows)
    private_context.Flush

    //show again
    if not Visible then
        me.height = height
    end
End Sub
EraseBackgroundGL As Boolean

```

## **RBGLCanvas.Graphics:**

Graphics As RBGLGraphics

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild

```

```

        #pragma StackOverflowChecking false

        return private_context.Graphics
    End Get
End Property
Private private_context As RBGLContext

StretchBackdrop As Boolean

End Class
Interface AbstractGraphics

    Sub Bold(assigns value as Boolean)

    End Sub
    Function Bold() As Boolean

    End Function
    Sub ClearRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)

    End Sub
    Function Clip(X As Integer, Y As Integer, Width As Integer, Height As Integer) As AbstractGraphics

    End Function
    Sub DrawCautionIcon(X As Integer, Y As Integer)

    End Sub
    Sub DrawLine(X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As Integer)

    End Sub
    Sub DrawNotelcon(X As Integer, Y As Integer)

    End Sub
    Sub DrawOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)

    End Sub
    Sub DrawPicture(image as Picture, x As Integer, y As Integer, w1 As Integer = - 10000, h1 As Integer = - 10000,
    sx As Integer = 0, sy As Integer = 0, w2 As Integer = - 10000, h2 As Integer = - 10000)

    End Sub
    Sub DrawPolygon(Points() As Single, ZeroBased As Boolean = False)

    End Sub
    Sub DrawPolygon(Points() As Double, ZeroBased As Boolean = False)

    End Sub
    Sub DrawPolygon(Points() As Integer, ZeroBased As Boolean = False)

    End Sub
    Sub DrawRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)

    End Sub
    Sub DrawRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight
    As Integer)

```

```

End Sub
Sub DrawStopIcon(X As Integer, Y As Integer)

End Sub
Sub DrawString(Text As String, X As Integer, Y As Integer, WrapWidth As Integer = 0, Condense As Boolean = False)

End Sub
Sub FillOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)

End Sub
Sub FillPolygon(Points() As Integer, ZeroBased As Boolean = False)

End Sub
Sub FillPolygon(Points() As Double, ZeroBased As Boolean = False)

End Sub
Sub FillPolygon(Points() As Single, ZeroBased As Boolean = False)

End Sub
Sub FillRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)

End Sub
Sub FillRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight As Integer)

End Sub
Sub ForeColor(assigns Col As Color)

End Sub
Function ForeColor() As Color

End Function
Function Height() As Integer

End Function
Function Italic() As Boolean

End Function
Sub Italic(assigns value as Boolean)

End Sub
Sub PenHeight(assigns value as Integer)

End Sub
Function PenHeight() As Integer

End Function
Sub PenWidth(assigns value as Integer)

End Sub
Function PenWidth() As Integer

End Function

```



Sub Pixel(X As Integer, Y As Integer, Assigns Col As Color)

End Sub

Function Pixel(X As Integer, Y As Integer) As Color

End Function

Function StringDirection(Text As String) As Integer

End Function

Function StringHeight(Text As String, WrapWidth As Integer) As Integer

End Function

Function StringWidth(Text As String) As Double

End Function

Function TextAscent() As Integer

End Function

Function TextFont() As String

End Function

Sub TextFont(assigns value as String)

End Sub

Function TextHeight() As Integer

End Function

Function TextSize() As Integer

End Function

Function Underline() As Boolean

End Function

Sub Underline(assigns value as Boolean)

End Sub

Function UseOldRenderer() As Boolean

End Function

Sub UseOldRenderer(assigns value as Boolean)

End Sub

Function Width() As Integer

End Function

End Interface

Class GraphicsWrapper

Implements AbstractGraphics

### **GraphicsWrapper.Bold:**

Sub Bold(assigns value as Boolean)

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

```
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
private_graphics.Bold = value
End Sub
```

### **GraphicsWrapper.Bold:**

```
Function Bold() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.Bold
End Function
```

### **GraphicsWrapper.ClearRect:**

```
Sub ClearRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.ClearRect x, y, Width, Height
End Sub
```

### **GraphicsWrapper.Clip:**

```
Function Clip(X As Integer, Y As Integer, Width As Integer, Height As Integer) As AbstractGraphics
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return new GraphicsWrapper(private_graphics.Clip(x,y,Width,Height))
End Function
```

### **GraphicsWrapper.Constructor:**

```
Sub Constructor(g as Graphics)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics = g
End Sub
```

### **GraphicsWrapper.DrawCautionIcon:**

```
Sub DrawCautionIcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
End Sub
```

```
private_graphics.DrawCautionIcon X, Y
End Sub
```

### **GraphicsWrapper.DrawLine:**

```
Sub DrawLine(X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawLine X1, Y1, X2, Y2
End Sub
```

### **GraphicsWrapper.DrawNotelcon:**

```
Sub DrawNotelcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawNotelcon x, y
End Sub
```

### **GraphicsWrapper.DrawOval:**

```
Sub DrawOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawOval x, y, width, Height
End Sub
```

### **GraphicsWrapper.DrawPicture:**

```
Sub DrawPicture(image as Picture, x As Integer, y As Integer, w1 As Integer = - 10000, h1 As Integer = - 10000,
sx As Integer = 0, sy As Integer = 0, w2 As Integer = - 10000, h2 As Integer = - 10000)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawPicture image, x, y, w1, h1, sx, sy, w2, h2
End Sub
```

### **GraphicsWrapper.DrawPolygon:**

```
Sub DrawPolygon(Points() As Single, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    dim p() as Integer
```

```

    redim p(ubound(points))
    for i as integer = ubound(Points) downto 0
        p(i) = Points(i)
    next

    if ZeroBased then
        p.Insert 0, 0
    end

    private_graphics.DrawPolygon p
End Sub

```

### **GraphicsWrapper.DrawPolygon:**

```

Sub DrawPolygon(Points() As Integer, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if ZeroBased then

        Points.Insert 0, 0
        private_graphics.DrawPolygon Points
        Points.Remove 0

    else

        private_graphics.DrawPolygon Points

    end
End Sub

```

### **GraphicsWrapper.DrawPolygon:**

```

Sub DrawPolygon(Points() As Double, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    dim p() as Integer
    redim p(ubound(points))
    for i as integer = ubound(Points) downto 0
        p(i) = Points(i)
    next

    if ZeroBased then
        p.Insert 0, 0
    end

    private_graphics.DrawPolygon p
End Sub

```

### **GraphicsWrapper.DrawRect:**

```

Sub DrawRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawRect x, y, Width, Height
End Sub

```

### **GraphicsWrapper.DrawRoundRect:**

```

Sub DrawRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawRoundRect x, y, Width, Height, ArcWidth, ArcHeight
End Sub

```

### **GraphicsWrapper.DrawStopIcon:**

```

Sub DrawStopIcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawStopIcon x, y
End Sub

```

### **GraphicsWrapper.DrawString:**

```

Sub DrawString(Text As String, X As Integer, Y As Integer, WrapWidth As Integer = 0, Condense As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.DrawString text, x, y, WrapWidth, Condense
End Sub

```

### **GraphicsWrapper.FillOval:**

```

Sub FillOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.FillOval x, y, Width, Height
End Sub

```

### **GraphicsWrapper.FillPolygon:**

```

Sub FillPolygon(Points() As Integer, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false

```

```

#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

if ZeroBased then

    Points.Insert 0, 0
    private_graphics.FillPolygon Points
    Points.Remove 0

else

    private_graphics.FillPolygon Points

end
End Sub

```

### **GraphicsWrapper.FillPolygon:**

```

Sub FillPolygon(Points() As Double, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    dim p() as Integer
    redim p(ubound(points))
    for i as integer = ubound(Points) downto 0
        p(i) = Points(i)
    next

    if ZeroBased then
        p.Insert 0, 0
    end

    private_graphics.FillPolygon p
End Sub

```

### **GraphicsWrapper.FillPolygon:**

```

Sub FillPolygon(Points() As Single, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    dim p() as Integer
    redim p(ubound(points))
    for i as integer = ubound(Points) downto 0
        p(i) = Points(i)
    next

    if ZeroBased then
        p.Insert 0, 0
    end

```

```
private_graphics.FillPolygon p
End Sub
```

### **GraphicsWrapper.FillRect:**

```
Sub FillRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.FillRect x, y, Width, Height
End Sub
```

### **GraphicsWrapper.FillRoundRect:**

```
Sub FillRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.FillRoundRect x, y, width, Height, ArcWidth, ArcHeight
End Sub
```

### **GraphicsWrapper.ForeColor:**

```
Function ForeColor() As Color
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.ForeColor
End Function
```

### **GraphicsWrapper.ForeColor:**

```
Sub ForeColor(Assigns Col As Color)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.ForeColor = col
End Sub
```

### **GraphicsWrapper.Height:**

```
Function Height() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.Height
End Function
```

### **GraphicsWrapper.Italic:**

```
Function Italic() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.Italic
End Function
```

### **GraphicsWrapper.Italic:**

```
Sub Italic(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.Italic = value
End Sub
```

### **GraphicsWrapper.PenHeight:**

```
Sub PenHeight(assigns value as Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.PenHeight = value
End Sub
```

### **GraphicsWrapper.PenHeight:**

```
Function PenHeight() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.PenHeight
End Function
```

### **GraphicsWrapper.PenWidth:**

```
Sub PenWidth(assigns value as Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.PenWidth = value
End Sub
```

### **GraphicsWrapper.PenWidth:**

```
Function PenWidth() As Integer
```



```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.PenWidth
End Function
```

### **GraphicsWrapper.Pixel:**

```
Function Pixel(X As Integer, Y As Integer) As Color
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
return private_graphics.pixel(x,y)
End Function
```

### **GraphicsWrapper.Pixel:**

```
Sub Pixel(X As Integer, Y As Integer, Assigns Col As Color)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
private_graphics.pixel(x,y) = col
End Sub
```

### **GraphicsWrapper.StringDirection:**

```
Function StringDirection(Text As String) As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
return private_graphics.StringDirection(text)
End Function
```

### **GraphicsWrapper.StringHeight:**

```
Function StringHeight(Text As String, WrapWidth As Integer) As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
return private_graphics.StringHeight(text, WrapWidth)
End Function
```

### **GraphicsWrapper.StringWidth:**

```
Function StringWidth(Text As String) As Double
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_graphics.StringWidth(text)
```

```
End Function
```

### **GraphicsWrapper.TextAscent:**

```
Function TextAscent() As Integer
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_graphics.TextAscent
```

```
End Function
```

### **GraphicsWrapper.TextFont:**

```
Function TextFont() As String
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_graphics.TextFont
```

```
End Function
```

### **GraphicsWrapper.TextFont:**

```
Sub TextFont(assigns value as String)
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
private_graphics.TextFont = value
```

```
End Sub
```

### **GraphicsWrapper.TextHeight:**

```
Function TextHeight() As Integer
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_graphics.TextHeight
```

```
End Function
```

### **GraphicsWrapper.TextSize:**

```
Function TextSize() As Integer
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_graphics.TextSize
```

End Function

### **GraphicsWrapper.Underline:**

```
Function Underline() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.Underline
End Function
```

### **GraphicsWrapper.Underline:**

```
Sub Underline(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.Underline = value
End Sub
```

### **GraphicsWrapper.UseOldRenderer:**

```
Function UseOldRenderer() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.UseOldRenderer
End Function
```

### **GraphicsWrapper.UseOldRenderer:**

```
Sub UseOldRenderer(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_graphics.UseOldRenderer = value
End Sub
```

### **GraphicsWrapper.Width:**

```
Function Width() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_graphics.Width
End Function

Private private_graphics As Graphics
```

End Class

Class RBGLContext

```
Private Const AGL_ACCELERATED = 73
Private Const AGL_ACCUM_ALPHA_SIZE = 17
Private Const AGL_ACCUM_BLUE_SIZE = 16
Private Const AGL_ACCUM_GREEN_SIZE = 15
Private Const AGL_ACCUM_RED_SIZE = 14
Private Const AGL_BUFFER_RECT = 202
Private Const AGL_DEPTH_SIZE = 12
Private Const AGL_NONE = 0
Private Const AGL_OFFSCREEN = 53
Private Const AGL_PBUFFER = 90
Private Const AGL_RGBA = 4
Private Const AGL_SAMPLES_ARB = 56
Private Const AGL_SAMPLE_BUFFERS_ARB = 55
Private Const AGL_STENCIL_SIZE = 13
Private Const AGL_SWAP_INTERVAL = 222
Private Const AGL_WINDOW = 80
Private Const GL_BACK = 1029
Private Const GL_BLEND = 3042
Private Const GL_COLOR_BUFFER_BIT = 16384
Private Const GL_DEPTH_BUFFER_BIT = 256
Private Const GL_EXTENSIONS = &h1F03
Private Const GL_MODELVIEW = 5888
Private Const GL_MULTISAMPLE_ARB = 32925
Private Const GL_ONE_MINUS_SRC_ALPHA = 771
Private Const GL_PROJECTION = 5889
Private Const GL_SCISSOR_TEST = 3089
Private Const GL_SRC_ALPHA = 770
Private Const GL_TEXTURE_2D = 3553
Private Const GL_TRUE = 1
Private Const PFD_DOUBLEBUFFER = 1
Private Const PFD_DRAW_TO_WINDOW = 4
Private Const PFD_MAIN_PLANE = 0
Private Const PFD_SUPPORT_OPENGL = 32
Private Const PFD_TYPE_RGBA = 0
Private Const WGL_ACCELERATION_ARB = &h2003
Private Const WGL_ALPHA_BITS_ARB = &h201B
Private Const WGL_BLUE_BITS_ARB = 8217
Private Const WGL_COLOR_BITS_ARB = &h2014
Private Const WGL_DEPTH_BITS_ARB = &h2022
Private Const WGL_DOUBLE_BUFFER_ARB = &h2011
Private Const WGL_DRAW_TO_PBUFFER_ARB = 8237
Private Const WGL_DRAW_TO_WINDOW_ARB = &h2001
Private Const WGL_FULL_ACCELERATION_ARB = &h2027
Private Const WGL_GREEN_BITS_ARB = 8215
Private Const WGL_RED_BITS_ARB = 8213
Private Const WGL_SAMPLES_ARB = &h2042
Private Const WGL_SAMPLE_BUFFERS_ARB = &h2041
Private Const WGL_STENCIL_BITS_ARB = &h2023
Private Const WGL_SUPPORT_OPENGL_ARB = &h2010
```

**RBGLContext.Bind:**

Sub Bind()

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//ignore if already current context
if private_current_context <> nil and private_current_context.value = me then
```

```
    //determine current context
    dim current as uint32
    #if TargetCarbon
        declare function aglGetCurrentContext lib AGL_LIB () as uint32
        current = aglGetCurrentContext
    #elseif TargetWin32
        declare function wglGetCurrentDC lib GL_LIB () as uint32
        current = wglGetCurrentDC
    #endif

    if current = private_handle then
        return
    end
```

end

```
#if TargetCarbon
```

```
    //set this context to be the current one
    declare function aglSetCurrentContext lib AGL_LIB (ctx as uint32) as boolean
    if not aglSetCurrentContext(private_handle) then raise new RBGLException("Unable to bind context")
```

```
    //if context is a pixel buffer then bind pBuffer object
    if private_pbuffer > 0 then
```

```
        //get virtual screen of current context, or use zero if no context set
        dim screen as uint32 = 0
        if private_current_context <> nil and private_current_context.value <> nil then
            dim prevctx as RBGLContext = RBGLContext(private_current_context.value)
            declare function aglGetVirtualScreen lib AGL_LIB (ctx as uint32) as int32
            screen = aglGetVirtualScreen(prevctx.Handle)
        end
```

```
        //bind pBuffer
        declare function aglSetPBuffer lib AGL_LIB (ctx as uint32, pBuffer as uint32, face as uint32, level as uint32,
            screen as uint32) as boolean
        call aglSetPBuffer private_handle, private_pbuffer, 0, 0, screen
```

end

```
#elseif TargetWin32
```

```
    //set this context to be the current one
    declare sub wglMakeCurrent lib GL_LIB (hdc as uint32, hglrc as uint32)
    wglMakeCurrent private_handle, private_hglrc
```

```

//use back buffer for drawing
declare sub glDrawBuffer lib GL_LIB (mode as uint32)
glDrawBuffer GL_BACK

//use back buffer for reading
declare sub glReadBuffer lib GL_LIB (mode as uint32)
glReadBuffer(GL_BACK)

#else

    raise new RBGLException("Unsupported platform")

#endif

//cleanup deleted textures
if private_deleted_textures_by_context <> nil and private_deleted_textures_by_context.haskey(private_handle)
then

    //context is current – delete immediately
    declare sub glDeleteTextures lib GL_LIB (n as uint32, textures as Ptr)
    dim ids as MemoryBlock = MemoryBlock(private_deleted_textures_by_context.value(private_handle))
    glDeleteTextures ids.Size\4, ids

    //remove me from dictionary
    private_deleted_textures_by_context.remove(private_handle)

end

//bind previous texture
if private_texture <> nil then

    //enable texture mapping
    declare sub glEnable lib GL_LIB (cap as uint32)
    glEnable GL_TEXTURE_2D

    //rebind texture
    private_texture.Bind

else

    //disable texture mapping
    declare sub glDisable lib GL_LIB (cap as uint32)
    glDisable GL_TEXTURE_2D

end

//set clear color
declare sub glClearColor lib GL_LIB (red as single, green as single, blue as single, alpha as single)
glClearColor private_clear_color.red/255.0, private_clear_color.green/255.0, _
private_clear_color.blue/255.0, private_clear_color.Alpha/255.0

//enable and disable features
for i as integer = private_features.Count – 1 downto 0

```

```

dim feature as UInt32 = private_features.Key(i).UInt32Value
if private_features.Value(feature) then

    //enable feature
    declare sub glEnable lib GL_LIB (cap as UInt32)
    glEnable feature

else

    //disable feature
    declare sub glDisable lib GL_LIB (cap as UInt32)
    glDisable feature
end
next

//set this as current context
private_current_context = new WeakRef(me)

```

End Sub

### **RBGLContext.Clear:**

```

Sub Clear(mask as Int32 = -10000)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //clear view
    declare sub glClear lib GL_LIB (mask as uint32)
    if mask = -10000 then mask = GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
    glClear mask
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Canvas As Canvas, options As RBGLContextOptions)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set reference
    private_window_or_canvas = new WeakRef(Canvas)

    #if TargetCarbon

        //create context
        Constructor Canvas.Window.Handle, options

        //enable buffer rect
        declare function aglEnable lib AGL_LIB (ctx as uint32, pname as UInt32) as Boolean
        if not aglEnable (private_handle, AGL_BUFFER_RECT) then raise new RBGLException("Failed to enable AGL
        buffer rect")

        //enable display sync
    
```

```

        call aglEnable (private_handle, AGL_SWAP_INTERVAL)

    #else

        //create context
        Constructor Canvas.Handle, options

    #endif
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Window As Window, options As RBGLContextOptions)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set reference
    private_window_or_canvas = new WeakRef(Window)

    //create context
    Constructor Window.Handle, options

    #if TargetCarbon

        //enable display sync
        declare function aglEnable lib AGL_LIB (ctx as uint32, pname as UInt32) as Boolean
        call aglEnable (private_handle, AGL_SWAP_INTERVAL)

    #endif
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Width as Integer, Height as Integer, options as RBGLContextOptions)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    #if TargetCarbon

        //define pixel format
        dim attribs as new MemoryBlock(80)
        attribs.uint32value(0) = AGL_RGBA
        attribs.uint32value(4) = AGL_DEPTH_SIZE
        attribs.uint32value(8) = options.DepthBufferBits
        attribs.uint32value(12) = AGL_STENCIL_SIZE
        attribs.uint32value(16) = options.StencilBufferBits
        attribs.uint32value(20) = AGL_ACCUM_RED_SIZE
        attribs.uint32value(24) = options.AccumulationBufferBits \ 4
        attribs.uint32value(28) = AGL_ACCUM_GREEN_SIZE
        attribs.uint32value(32) = options.AccumulationBufferBits \ 4
        attribs.uint32value(36) = AGL_ACCUM_BLUE_SIZE
    #endif
End Sub

```



```

attribs.uint32value(40) = options.AccumulationBufferBits \ 4
attribs.uint32value(44) = AGL_ACCUM_ALPHA_SIZE
attribs.uint32value(48) = options.AccumulationBufferBits \ 4
attribs.uint32value(52) = AGL_PBUFFER
attribs.uint32value(56) = AGL_OFFSCREEN
if options.Oversampling > 0 then
    attribs.uint32value(60) = AGL_SAMPLE_BUFFERS_ARB
    attribs.uint32value(64) = 1
    attribs.uint32value(68) = AGL_SAMPLES_ARB
    attribs.uint32value(72) = options.Oversampling
    attribs.uint32value(76) = AGL_NONE
else
    attribs.uint32value(60) = AGL_NONE
end

//get the pixel format
dim fmt as uint32
if System.IsFunctionAvailable("aglCreatePixelFormat", AGL_LIB) then

    //use new CreatePixelFormat function
    soft declare function aglCreatePixelFormat lib AGL_LIB (attribs as ptr) as uint32
    fmt = aglCreatePixelFormat(attribs)

else

    //choose pixel format
    soft declare function aglChoosePixelFormat lib AGL_LIB (gdevs as uint32, ndev as uint32, attribs as ptr) as
    uint32
    fmt = aglChoosePixelFormat(0,0,attribs)

end

//create the context
declare function aglCreateContext lib AGL_LIB (pix as uint32, share as uint32) as integer
private_handle = aglCreateContext(fmt, 0)

//round dimensions to nearest power of two
private_pbuffer_width = Private_round_to_power_of_two(width)
private_pbuffer_height = Private_round_to_power_of_two(height)

//create a pixel buffer object
declare function aglCreatePBuffer lib AGL_LIB (private_pbuffer_width as integer, private_pbuffer_height as
integer, target as integer, internalFormat as RBGLPixelFormat, max_level as integer, byref pbuffer as uint32)
as Boolean
call aglCreatePBuffer(width, height, GL_TEXTURE_2D, RBGLPixelFormat.RGBA, 0, private_pbuffer)

#elseif TargetWin32

    //get current hdc
    dim current_hdc as uint32
    #if DebugBuild
        if private_current_context <> nil and private_current_context.value <> nil then
            current_hdc = RBGLContext(private_current_context.value).Handle
        else

```

```

        raise new RBGLEException("Cannot create a pbuffer before a window context has been bound")
    end
#endif

//define pixel format
dim attribs as new MemoryBlock(72)
attribs.uint32value(0) = WGL_SUPPORT_OPENGL_ARB
attribs.uint32value(4) = 1
attribs.uint32value(8) = WGL_DRAW_TO_PBUFFER_ARB
attribs.uint32value(12) = 1
attribs.uint32value(16) = WGL_RED_BITS_ARB
attribs.uint32value(24) = 8
attribs.uint32value(28) = WGL_GREEN_BITS_ARB
attribs.uint32value(32) = 8
attribs.uint32value(36) = WGL_BLUE_BITS_ARB
attribs.uint32value(40) = 8
attribs.uint32value(44) = WGL_ALPHA_BITS_ARB
attribs.uint32value(48) = 8
attribs.uint32value(52) = WGL_DEPTH_BITS_ARB
attribs.uint32value(56) = options.DepthBufferBits
attribs.uint32value(60) = WGL_DOUBLE_BUFFER_ARB
attribs.uint32value(64) = 0
attribs.uint32value(68) = 0

//get pointer to pbuffer create function
declare function wglGetProcAddress lib GL_LIB (lpszProc as CString) as Ptr
dim fnPtr as Ptr = wglGetProcAddress("wglChoosePixelFormatARB")

//choose pixel format
dim pixelFormat as uint32
if fnPtr <> nil then
    dim null as ptr
    dim count as uint32 = 0
    dim fn as new wglChoosePixelFormatARBDelegate(fnPtr)
    call fn.Invoke(current_hdc, attribs, null, 1, pixelFormat, count)
    if count = 0 then
        raise new RBGLEException("could not find a suitable pixel format for the pbuffer")
    end
else
    raise new RBGLEException("wglChoosePixelFormatARB is not supported on this system")
end

//get pointer to pbuffer create function
fnPtr = wglGetProcAddress("wglCreatePbufferARB")

//create pbuffer
if fnPtr <> nil then
    dim null as ptr
    dim fn as new wglCreatePbufferARBDelegate(fnPtr)
    private_pbuffer = fn.Invoke(private_handle, pixelFormat, private_pbuffer_width, private_pbuffer_height, null)
    if private_pbuffer = 0 then
        raise new RBGLEException("could not create pbuffer")
    end
end

```

```

else
    raise new RBGLException("wglCreatePbufferARB is not supported on this system")
end

//get pointer to hdc create function
fnPtr = wglGetProcAddress("wglGetPbufferDCARB")

//create hdc
if fnPtr <> nil then
    dim fn as new wglGetPbufferDCARBDelegate(fnPtr)
    private_handle = fn.Invoke(private_pbuffer)
    if private_handle = 0 then
        raise new RBGLException("could not create hdc for pbuffer")
    end
else
    raise new RBGLException("wglGetPbufferDCARB is not supported on this system")
end

//create context
declare function wglCreateContext lib GL_LIB (hdc as uint32) as uint32
private_hglrc = wglCreateContext(private_handle)

#else

    raise new RBGLException("Unsupported platform")

#endif

//initialise features dictionary
private_features = new Dictionary

//bind context
Bind

//enable default options
declare sub glEnable lib GL_LIB (cap as uint32)
glEnable GL_SCISSOR_TEST
glEnable GL_BLEND

//set default blend rule
declare sub glBlendFunc lib GL_LIB (sfactor as uint32, dfactor as uint32)
glBlendFunc GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA

//set viewport
UpdateDimensions
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Canvas As Canvas)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//create with default options
Constructor Canvas, DefaultOptions
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Window As Window)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//create with default options
Constructor Window, DefaultOptions
End Sub

```

### **RBGLContext.Constructor:**

```

Sub Constructor(Width as Integer, Height as Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//create with default options
Constructor width, height, DefaultOptions
End Sub

```

### **RBGLContext.Constructor:**

```

Private Sub Constructor(WindowOrCanvasPtr As UInt32, options as RBGLContextOptions)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

#if TargetCarbon

```

```

//define pixel format
dim attribs as new MemoryBlock(72)
attribs.uint32value(0) = AGL_RGBA
attribs.uint32value(4) = AGL_DEPTH_SIZE
attribs.uint32value(8) = options.DepthBufferBits
attribs.uint32value(12) = AGL_STENCIL_SIZE
attribs.uint32value(16) = options.StencilBufferBits
attribs.uint32value(20) = AGL_ACCUM_RED_SIZE
attribs.uint32value(24) = options.AccumulationBufferBits \ 4
attribs.uint32value(28) = AGL_ACCUM_GREEN_SIZE
attribs.uint32value(32) = options.AccumulationBufferBits \ 4
attribs.uint32value(36) = AGL_ACCUM_BLUE_SIZE
attribs.uint32value(40) = options.AccumulationBufferBits \ 4
attribs.uint32value(44) = AGL_ACCUM_ALPHA_SIZE
attribs.uint32value(48) = options.AccumulationBufferBits \ 4
if options.Oversampling > 0 then
    attribs.uint32value(52) = AGL_SAMPLE_BUFFERS_ARB
    attribs.uint32value(56) = 1

```

```

    attribs.uint32value(60) = AGL_SAMPLES_ARB
    attribs.uint32value(64) = options.Oversampling
    attribs.uint32value(68) = AGL_NONE
else
    attribs.uint32value(52) = AGL_NONE
end

//get the pixel format
dim fmt as uint32
if System.IsFunctionAvailable("aglCreatePixelFormat", AGL_LIB) then

    //use new CreatePixelFormat function
    soft declare function aglCreatePixelFormat lib AGL_LIB (attribs as ptr) as uint32
    fmt = aglCreatePixelFormat(attribs)

else

    //choose pixel format
    soft declare function aglChoosePixelFormat lib AGL_LIB (gdevs as uint32, ndev as uint32, attribs as ptr) as
    uint32
    fmt = aglChoosePixelFormat(0,0,attribs)

end

//create the context
declare function aglCreateContext lib AGL_LIB (pix as uint32, share as uint32) as integer
private_handle = aglCreateContext(fmt, 0)

if System.IsFunctionAvailable("aglSetWindowRef",AGL_LIB) then
    //use new window ref function

    //attach the context to the window
    soft declare function aglSetWindowRef lib AGL_LIB (ctx as uint32, window as uint32) as boolean
    if not aglSetWindowRef(private_handle, WindowOrCanvasPtr) then raise new RBGLException("Failed to
    attach context to window")

else
    //use deprecated grafport reference

    //get the graphics port associated with the window
    soft declare function GetWindowPort Lib CARBON_LIB (window as uint32) as integer
    dim port as uint32 = GetWindowPort(WindowOrCanvasPtr)

    //attach the context to the port
    soft declare function aglSetDrawable lib AGL_LIB (ctx as uint32, draw as uint32) as boolean
    if not aglSetDrawable(private_handle, port) then raise new RBGLException("Failed to attach context to
    window port")

end

//destroy pixel format
declare sub aglDestroyPixelFormat lib AGL_LIB (pix as integer)
aglDestroyPixelFormat fmt

```

```
#elseif TargetWin32
```

```
//define pixel format
dim size as uint32 = 26*4
dim descriptor as new MemoryBlock(size)
descriptor.uint16value(0) = size
descriptor.uint16value(4) = 1 //version number
descriptor.uint32value(8) = PFD_DRAW_TO_WINDOW + PFD_SUPPORT_OPENGL + PFD_DOUBLEBUFFER
descriptor.uint8value(12) = PFD_TYPE_RGBA
descriptor.uint8value(16) = 24 //colour bit depth
descriptor.uint8value(20) = 0 //red bits
descriptor.uint8value(24) = 0 //red shift
descriptor.uint8value(28) = 0 //green bits
descriptor.uint8value(32) = 0 //green shift
descriptor.uint8value(36) = 0 //blue bits
descriptor.uint8value(40) = 0 //blue shift
descriptor.uint8value(44) = 0 //alpha bits
descriptor.uint8value(48) = 0 //alpha shift
descriptor.uint8value(52) = options.AccumulationBufferBits //accumulation buffer bits
descriptor.uint8value(56) = 0 //accumulation buffer red bits
descriptor.uint8value(60) = 0 //accumulation buffer green bits
descriptor.uint8value(64) = 0 //accumulation buffer blue bits
descriptor.uint8value(68) = 0 //accumulation buffer alpha bits
descriptor.uint8value(72) = options.DepthBufferBits //depth buffer bits
descriptor.uint8value(76) = options.StencilBufferBits //stencil buffer bits
descriptor.uint8value(80) = 0 //auxiliary buffers (not supported)
descriptor.uint32value(84) = PFD_MAIN_PLANE //main layer (ignored)
descriptor.uint32value(88) = 0 //number of underlay and overlay planes
descriptor.uint32value(92) = 0 //layer mask (ignored)
descriptor.uint32value(96) = 0 //visible mask
descriptor.uint32value(100) = 0 //damage mask

//create hdc
declare function GetDC lib WINUSER_LIB (hwnd as uint32) as uint32
private_handle = GetDC(WindowOrCanvasPtr)

//choose pixel format
declare function ChoosePixelFormat lib WINGDI_LIB (hdc as uint32, pixelformatdescriptor as ptr) as uint32
dim fmt as uint32 = ChoosePixelFormat(private_handle, descriptor)

//set pixel format
declare function SetPixelFormat lib WINGDI_LIB (hdc as uint32, fmt as uint32, pixelformatdescriptor as ptr) as Boolean
if not SetPixelFormat(private_handle, fmt, descriptor) then raise new RBGLException("Failed to set pixel format")

//create context
declare function wglCreateContext lib GL_LIB (hdc as uint32) as uint32
private_hglrc = wglCreateContext(private_handle)
```

```
#else
```

```
raise new RBGLException("Unsupported platform")
```

```

#endif

//initialise features dictionary
private_features = new Dictionary

//bind context
Bind

//enable default options
declare sub glEnable lib GL_LIB (cap as uint32)
glEnable GL_SCISSOR_TEST
glEnable GL_BLEND

//set default blend rule
declare sub glBlendFunc lib GL_LIB (sfactor as uint32, dfactor as uint32)
glBlendFunc GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA

//set the clear colour to match the parent window
ClearColor = RGBA(Window.BackColor,255)

//initialise viewport
UpdateDimensions
End Sub

```

### **RBGLContext.DefaultOptions:**

Shared Function DefaultOptions() As RBGLContextOptions

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

```

```

dim options as RBGLContextOptions
return options

```

End Function

### **RBGLContext.Destructor:**

Sub Destructor()

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

```

```

//detach context from window
#if TargetCarbon

```

```

    if private_pbuffer = 0 then
        if System.IsFunctionAvailable("aglSetHViewRef",AGL_LIB) then

```

```

            //use new window ref function
            soft declare function aglSetHViewRef lib AGL_LIB (ctx as uint32, hiviewref as uint32) as boolean
            if not aglSetHViewRef(private_handle, 0) then raise new RBGLException("Failed to detach context from
            window")

```

```

        else

```

```

        //use deprecated setdrawable function
        soft declare function aglSetDrawable lib AGL_LIB (ctx as uint32, draw as uint32) as boolean
        if not aglSetDrawable(private_handle, 0) then raise new RBGLException("Failed to detach context from
        window")

        end
    end

#endif

#if TargetCarbon

    //delete pbuffer object
    if private_pbuffer > 0 then
        declare sub aglDestroyPBuffer lib AGL_LIB (pbuffer as uint32)
        aglDestroyPBuffer private_pbuffer
        end

    //delete the agl context
    declare sub aglDestroyContext lib AGL_LIB (ctx as uint32)
    aglDestroyContext private_handle

#elseif TargetWin32

    //delete the wgl context
    declare function wglDeleteContext lib GL_LIB (hglrc as uint32) as Boolean
    if not wglDeleteContext(private_hglrc) then raise new RBGLException("Failed to delete context")

    //release the dc
    declare sub ReleaseDC lib WINUSER_LIB (hwnd as uint32, hdc as uint32)
    if private_window_or_canvas.Value isa Canvas then
        ReleaseDC Canvas(private_window_or_canvas.Value).Handle, private_handle
    else
        ReleaseDC Window(private_window_or_canvas.Value).Handle, private_handle
    end

#else

    raise new RBGLException("Unsupported platform")

#endif
End Sub

```

### **RBGLContext.Enable:**

```

Sub Enable(feature as RBGLFeature, enable as Boolean = True)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set value
    private_features.Value(UInt32(feature)) = enable

```



```

//set feature if bound
if private_current_context <> nil and private_current_context.value = me then
    if enable then

        //enable feature
        declare sub glEnable lib GL_LIB (cap as RBGLFeature)
        glEnable feature

    else

        //disable feature
        declare sub glDisable lib GL_LIB (cap as RBGLFeature)
        glDisable feature

    end
end
End Sub

```

### **RBGLContext.Flush:**

```

Sub Flush()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    #if TargetCarbon

        //flush pending commands to output buffer
        declare sub glFlush lib GL_LIB ()
        glFlush

    #elseif TargetWin32

        //swap front and back buffers
        declare function wglSwapBuffers lib GL_LIB (hdc as uint32) as Boolean
        if not wglSwapBuffers(private_handle) then raise new RBGLException("Unable to swap buffers")

    #else

        raise new RBGLException("Unsupported platform")

    #endif
End Sub

```

### **RBGLContext.GetRBGLPicture:**

```

Function GetRBGLPicture(X As Integer = 0, Y As Integer = 0, Width As Integer = -10000, Height As Integer = -10000) As RBGLPicture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//set default width/height
if Width = -10000 then width = me.Width
if Height = -10000 then height = me.Height

//get picture
return new RBGLPicture(me, X, Y, Width, Height, RBGLPicture.DefaultOptions)
End Function

```

### **RBGLContext.Handle:**

```

Function Handle() As UInt32
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //return handle
    return private_handle
End Function

```

### **RBGLContext.IsExtensionSupported:**

```

Function IsExtensionSupported(ExtensionName as String) As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    static supported as String
    if supported = "" then

        #if TargetWin32 and DebugBuild
            if private_current_context = nil or private_current_context.value = nil then
                raise new RBGLException("Cannot test for extensions before context has been bound")
            end
        #endif

        //get the extensions list
        declare function glGetString lib GL_LIB (pname as Int32) as CString
        supported = "" + glGetString(GL_EXTENSIONS) + " "

        #if TargetWin32

            //get pointer to extended list function
            declare function wglGetProcAddress lib GL_LIB (lpszProc as CString) as Ptr
            dim fnPtr as Ptr = wglGetProcAddress("wglGetExtensionsStringARB")

            //get extended list
            if fnPtr <> nil then
                dim fn as new wglGetExtensionsStringARBDelegate(fnPtr)
                supported = supported + fn.Invoke(private_handle) + " "
            end

        #endif
    end if
end Function

```

end

```
//match extension
return supported.InStr(" " + ExtensionName + " ") > 0
```

End Function

### **RBGLContext.LastError:**

```
Function LastError() As RBGLErrorCode
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //get opengl error
    declare function glGetError lib GL_LIB () as RBGLErrorCode
    return glGetError
End Function
```

### **RBGLContext.SetMatrixAuthographic:**

```
Sub SetMatrixAuthographic(Left as Double = -1.0, Top as Double = 1.0, Right as Double = 1.0, Bottom as Double
= -1.0, NearClip as Double = -500, FarClip as Double = 500)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //set the matrix to projection mode
    declare sub glMatrixMode lib GL_LIB (mode as integer)
    glMatrixMode GL_PROJECTION

    //reset matrix
    declare sub glLoadIdentity lib GL_LIB ()
    glLoadIdentity

    //set orthographic view
    declare sub glOrtho lib GL_LIB (Left as Double, Right as Double, Bottom as Double, Top as Double, Near as
Double, Far as Double)
    glOrtho Left, Right, Bottom, Top, NearClip, FarClip

    //set the matrix to model view mode
    glMatrixMode GL_MODELVIEW

    //reset matrix
    glLoadIdentity
End Sub
```

### **RBGLContext.SetMatrixPerspective:**

```
Sub SetMatrixPerspective(FovY as Double = 90, AspectRatio as Double = -10000, NearClip as Double = 0, FarClip
as Double = 1000)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //set the matrix to projection mode
    declare sub glMatrixMode lib GL_LIB (mode as integer)
    glMatrixMode GL_PROJECTION

    //reset matrix
    declare sub glLoadIdentity lib GL_LIB ()
    glLoadIdentity

    //set perspective view
    declare sub gluPerspective lib GLU_LIB (fovy as Double, aspect as Double, zNear as Double, zFar as Double)
    if AspectRatio = -10000 then AspectRatio = Width/Height
    gluPerspective FovY, AspectRatio, NearClip, FarClip

    //set the matrix to model view mode
    glMatrixMode GL_MODELVIEW

    //reset matrix
    glLoadIdentity
End Sub
```

### **RBGLContext.UnBind:**

```
Sub UnBind()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //ignore if not current context
    if private_current_context = nil or private_current_context.value <> me then
        return
    end

    //flush before switching
    Flush

    #if TargetCarbon

        //set context to nil
        declare function aglSetCurrentContext lib AGL_LIB (ctx as integer) as boolean
        call aglSetCurrentContext(0)

    #elseif TargetWin32
```

```

//set context to nil
declare sub wglMakeCurrent lib GL_LIB (hdc as uint32, hglrc as uint32)
wglMakeCurrent(0, 0)

#else

    raise new RBGLException("Unsupported platform")

#endif

//clear current context
private_current_context = nil
End Sub

```

## **RBGLContext.UpdateDimensions:**

```

Sub UpdateDimensions()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
        if private_window_or_canvas.Value isa Canvas then

            dim c as Canvas = Canvas(private_window_or_canvas.Value)
            if c.left <> private_rect.left or c.top <> private_rect.top or _
                c.Width <> private_rect.Width or c.Height <> private_rect.height then

                //resize view
                private_rect.left = c.left
                private_rect.Top = c.top
                private_rect.Width = c.width
                private_rect.Height = c.Height

                #if TargetCarbon

                    //update context
                    declare function aglUpdateContext lib AGL_LIB (ctx as uint32) as Boolean
                    if not aglUpdateContext(private_handle) then raise new RBGLException("Failed to update context")

                    //set buffer rect
                    dim r as GLintRect = private_rect
                    r.Top = c.window.height - c.height - c.top
                    if c.top < 0 then r.height = r.height + c.top
                    declare function aglSetInteger lib AGL_LIB (ctx as uint32, pname as UInt32, byref params as
                        GLintRect) as Boolean
                    if not aglSetInteger(private_handle, AGL_BUFFER_RECT, r) then raise new RBGLException("Failed to set
                        buffer rect")

                #endif

            end if
        end if
    end if
End Sub

```

```

end

else

    dim w as Window = Window(private_window_or_canvas.Value)
    if w.Width <> private_rect.Width or w.Height <> private_rect.height then

        //resize view
        private_rect.Width = w.width
        private_rect.Height = w.Height

        #if TargetCarbon

            //update context
            declare function aglUpdateContext lib AGL_LIB (ctx as uint32) as Boolean
            if not aglUpdateContext(private_handle) then raise new RBGLException("Failed to update context")

        #endif

    end

end

elseif private_pbuffer > 0 then

    //resize view
    private_rect.Width = private_pbuffer_width
    private_rect.Height = private_pbuffer_height

end

//set viewport
declare sub glViewport lib GL_LIB (x as Int32, y as Int32, width as Int32, height as Int32)
glViewport 0, 0, private_rect.Width, private_rect.Height

//set scissor rect
declare sub glScissor lib GL_LIB (x as Int32, y as Int32, width as Int32, height as Int32)
glScissor 0, 0, Width, Height
End Sub

```

## **RBGLContext.ClearColor:**

ClearColor As RBGLColor

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_clear_color

End Get
Set
    #pragma BackgroundTasks false

```

```
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//set the color
private_clear_color = value
```

```
//update context clear color if bound
if private_current_context <> nil and private_current_context.value = me then
```

```
    //set the clear colour
    declare sub glClearColor lib GL_LIB (red as single, green as single, blue as single, alpha as single)
    glClearColor value.red/255.0, value.green/255.0, value.blue/255.0, value.Alpha/255.0
```

```
end
```

```
End Set
```

```
End Property
```

## **RBGLContext.Graphics:**

Graphics As RBGLGraphics

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//bind context (make it current)
Bind
```

```
//update context size and position
UpdateDimensions
```

```
//return new graphics object
return new RBGLGraphics(me)
```

```
End Get
```

```
End Property
```

## **RBGLContext.Height:**

Height As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
    if private_window_or_canvas.Value isa Window then
        return Window(private_window_or_canvas.Value).Height
    else
        return Canvas(private_window_or_canvas.Value).Height
    end
elseif private_pbuffer > 0 then
```

```

        return private_pbuffer_width
    end
End Get
End Property

```

## **RBGLContext.Left:**

Left As Integer

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
        if private_window_or_canvas.Value isa Window then
            return 0
        else
            return Canvas(private_window_or_canvas.Value).Left
        end
    end
End Get
End Property
Private private_clear_color As RBGLColor

Private private_features As Dictionary

Private private_handle As uint32

Private private_hglrc As uint32

Private private_pbuffer As uint32

Private private_pbuffer_height As Integer

Private private_pbuffer_width As Integer

Private private_rect As GLIntRect

Private private_texture As RBGLPicture

Private private_window_or_canvas As WeakRef

```

## **RBGLContext.Texture:**

Texture As RBGLPicture

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_texture

```



End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//ignore if already bound
if private_texture = value then
    return
end

//set texture
private_texture = value

//bind texture if context is currently bound
if private_current_context <> nil and private_current_context.value = me then

    if private_texture <> nil then

        //enable texture mapping
        declare sub glEnable lib GL_LIB (cap as uint32)
        glEnable GL_TEXTURE_2D

        //bind texture
        private_texture.Bind

    else

        //disable texture mapping
        declare sub glDisable lib GL_LIB (cap as uint32)
        glDisable GL_TEXTURE_2D

    end

end

end
End Set
End Property
```

## **RBGLContext.Top:**

Top As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
    if private_window_or_canvas.Value isa Window then
        return 0
    else
        return Canvas(private_window_or_canvas.Value).Top
    end
end
```

```
end
End Get
End Property
```

### **RBGLContext.Width:**

Width As Integer

```
Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
        if private_window_or_canvas.Value isa Window then
            return Window(private_window_or_canvas.Value).Width
        else
            return Canvas(private_window_or_canvas.Value).Width
        end
    elseif private_pbuffer > 0 then
        return private_pbuffer_width
    end
End Get
End Property
```

### **RBGLContext.Window:**

Window As Window

```
Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_window_or_canvas <> nil and private_window_or_canvas.Value <> nil then
        if private_window_or_canvas.Value isa Window then
            return Window(private_window_or_canvas.Value)
        else
            return Canvas(private_window_or_canvas.Value).Window
        end
    end
End Get
End Property
End Class
Class RBGLGraphics
Implements AbstractGraphics
```

```
Private Const CAUTION_ICON = 0
Private Const GL_FILL = &h1B02
Private Const GL_FRONT_AND_BACK = &h0408
Const GL_LINE = &h1B01
Private Const GL_LINES = &h0001
Private Const GL_LINE_LOOP = &h0002
Private Const GL_POLYGON = 9
```

```

Private Const GL_QUADS = 7
Private Const GL_QUAD_STRIP = 8
Private Const GL_RGBA = 6408
Private Const GL_UNSIGNED_BYTE = 5121
Private Const HALF_PI = 1.57079633
Private Const MAX_CIRCLE_EDGE_LENGTH = 4
Private Const MIN_CIRCLE_EDGES = 16
Private Const NOTE_ICON = 1
Private Const PI = 3.14159265
Private Const STOP_ICON = 2
Private Const TWO_PI = 6.28318531

```

### **RBGLGraphics.Bind:**

```

Sub Bind()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //check if already bound
    if private_current_graphics <> nil and private_current_graphics.value = me then
        return
    end

    //bind context
    private_context.Bind

    //set scissor rect
    declare sub glScissor lib GL_LIB (x as Int32, y as Int32, width as Int32, height as Int32)
    glScissor private_rect.Left, private_context.Height - private_rect.Top - private_rect.Height, private_rect.Width,
    private_rect.Height

    //set viewport
    declare sub glViewport lib GL_LIB (x as Int32, y as Int32, width as Int32, height as Int32)
    glViewport private_rect.Left, private_context.Height - private_rect.Top - private_rect.Height, private_rect.Width,
    private_rect.Height

    //reset matrix
    private_context.SetMatrixAuthographic 0,0,Width,Height

    //set colour
    declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
    glColor4ubv private_color

    //update current graphics
    private_current_graphics = new WeakRef(me)
End Sub

```

### **RBGLGraphics.Bold:**

```

Function Bold() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```
return private_string.Bold
```

```
End Function
```

### **RBGLGraphics.Bold:**

```
Sub Bold(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_string.Bold = value
```

```
End Sub
```

### **RBGLGraphics.Cache:**

```
Shared Sub Cache(Image As Picture)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //initialise cache dictionary
    if private_cache = nil then
        private_cache = new Dictionary
    end

    //cache image as a texture
    dim tex as new RBGLPicture(Image)
    tex.WrapX = RBGLWrapMode.Clamp
    tex.WrapY = RBGLWrapMode.Clamp
    private_cache.Value(Image) = tex
End Sub
```

### **RBGLGraphics.ClearCache:**

```
Shared Sub ClearCache()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_cache = new Dictionary
End Sub
```

### **RBGLGraphics.ClearRect:**

```
Sub ClearRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```

//bind me
Bind

//verify that selected region is within bounds
if x < private_rect.Width and y < private_rect.Height then

    //get graphics offset
    dim left as UInt32 = private_rect.Left
    dim top as UInt32 = private_context.Height - private_rect.Top

    //set scissor rect
    declare sub glScissor lib GL_LIB (x as Int32, y as Int32, width as Int32, height as Int32)
    glScissor left + x, top - y - Height, min(Width,private_rect.Width - x), min(Height,private_rect.Height - y)

    //clear region
    private_context.Clear

    //restore scissor rect
    glScissor left, top - private_rect.Height, private_rect.Width, private_rect.Height

end
End Sub

```

### **RBGLGraphics.Clip:**

```

Function Clip(X As Integer, Y As Integer, Width As Integer, Height As Integer) As RBGLGraphics
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //create new graphics
    return new RBGLGraphics( _
        private_context, _
        private_rect.Left + X, _
        private_rect.Top + Y, _
        min(width, private_rect.Width - private_rect.Left), _
        min(height, private_rect.Height - private_rect.Top))
End Function

```

### **RBGLGraphics.Constructor:**

```

Sub Constructor(Context as RBGLContext)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set context
    private_context = Context

    //set dimensions
    private_rect.Left = 0
    private_rect.Top = 0
    private_rect.Width = Context.Width
    private_rect.Height = Context.Height

```

```

//set default values
private_color.Alpha = 255

//create string texture
private_string = new RBGLStringTexture("")

//initialise cache
if private_cache = nil then
    private_cache = new Dictionary
end

//bind
Bind
End Sub

```

### **RBGLGraphics.Constructor:**

```

Sub Constructor(Context as RBGLContext, X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set context
    private_context = Context

    //set dimensions
    private_rect.Left = X
    private_rect.Top = Y
    private_rect.Width = Width
    private_rect.Height = Height

    //set default values
    private_color.Alpha = 255

    //create string texture
    private_string = new RBGLStringTexture("")

    //initialise cache
    if private_cache = nil then
        private_cache = new Dictionary
    end

    //bind
    Bind
End Sub

```

### **RBGLGraphics.DrawCautionIcon:**

```

Sub DrawCautionIcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//call other drawcautionicon method
DrawCautionIcon X, Y, False
End Sub

```

### **RBGLGraphics.DrawCautionIcon:**

```

Sub DrawCautionIcon(X As Integer, Y As Integer, Blend as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //build texture
    private_build_icon_texture CAUTION_ICON

    //draw icon
    DrawPicture private_icons(CAUTION_ICON), X, Y, Blend

End Sub

```

### **RBGLGraphics.DrawLine:**

```

Sub DrawLine(X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //clear texture
    private_context.Texture = nil

    if x1 = x2 or y1 = y2 then

        //draw rect
        declare sub glRecti lib GL_LIB (x1 as Integer, y1 as Integer, x2 as Integer, y2 as Integer)
        glRecti min(x1,x2), min(y1,y2), max(x1,x2)+private_penwidth, max(y1,y2)+private_penheight

    elseif private_pixel_pen then

        //begin line
        declare sub glBegin lib GL_LIB (mode as Uint32)
        glBegin GL_LINES

        //draw vertices
        declare sub glVertex2i lib GL_LIB (x as Integer, y as Integer)
        glVertex2i x1 - 0.5, y1 - 0.5
        glVertex2i x2 - 0.5, y2 - 0.5

        //end line
        declare sub glEnd lib GL_LIB ()

```

```
glEnd
```

```
else
```

```
//begin polygon  
declare sub glBegin lib GL_LIB (mode as UInt32)  
glBegin GL_POLYGON
```

```
//draw vertices  
declare sub glVertex2i lib GL_LIB (x as Integer, y as Integer)
```

```
//add pen width and height to dimensions  
dim x1PlusPenWidth as Integer = x1 + private_penwidth  
dim x2PlusPenWidth as Integer = x2 + private_penwidth  
dim y1PlusPenHeight as Integer = y1 + private_penheight  
dim y2PlusPenHeight as Integer = y2 + private_penheight
```

```
if x2 > x1 then  
  if y2 > y1 then
```

```
    glVertex2i x1, y1  
    glVertex2i x1PlusPenWidth, y1  
    glVertex2i x2PlusPenWidth, y2  
    glVertex2i x2PlusPenWidth, y2PlusPenHeight  
    glVertex2i x2, y2PlusPenHeight  
    glVertex2i x1, y1PlusPenHeight
```

```
  else
```

```
    glVertex2i x1, y1  
    glVertex2i x2, y2  
    glVertex2i x2PlusPenWidth, y2  
    glVertex2i x2PlusPenWidth, y2PlusPenHeight  
    glVertex2i x1PlusPenWidth, y1PlusPenHeight  
    glVertex2i x1, y1PlusPenHeight
```

```
  end
```

```
else  
  if y2 > y1 then
```

```
    glVertex2i x1, y1  
    glVertex2i x1PlusPenWidth, y1  
    glVertex2i x1PlusPenWidth, y1PlusPenHeight  
    glVertex2i x2PlusPenWidth, y2PlusPenHeight  
    glVertex2i x2, y2PlusPenHeight  
    glVertex2i x2, y2
```

```
  else
```

```
    glVertex2i x1PlusPenWidth, y1  
    glVertex2i x1PlusPenWidth, y1PlusPenHeight  
    glVertex2i x1, y1PlusPenHeight  
    glVertex2i x2, y2PlusPenHeight  
    glVertex2i x2, y2
```



```

        glVertex2i x2PlusPenWidth, y2

    end
end

//end polygon
declare sub glEnd lib GL_LIB ()
glEnd

end

End Sub

```

### **RBGLGraphics.DrawNotelcon:**

```

Sub DrawNotelcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //call other drawnoteicon method
    DrawNotelcon X, Y, False
End Sub

```

### **RBGLGraphics.DrawNotelcon:**

```

Sub DrawNotelcon(X As Integer, Y As Integer, Blend as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //build texture
    private_build_icon_texture NOTE_ICON

    //draw icon
    DrawPicture private_icons(NOTE_ICON), X, Y, Blend
End Sub

```

### **RBGLGraphics.DrawOval:**

```

Sub DrawOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //clear texture
    private_context.Texture = nil

```

```

//steps
dim radius as Single = (width+height)/2
dim circumference as Single = TWO_PI*radius
dim steps as Integer = max(MIN_CIRCLE_EDGES,circumference/MAX_CIRCLE_EDGE_LENGTH)
dim stp as Single = TWO_PI/steps

//centre and radius
dim radiusX as Single = width/2
dim centreX as Single = x + radiusX
dim radiusY as Single = height/2
dim centreY as Single = y + radiusY

if private_pixel_pen then

    //adjust radius
    radiusX = radiusX - 0.5
    radiusY = radiusY - 0.5

    //begin quad strip
    declare sub glBegin lib GL_LIB (mode as UInt32)
    glBegin GL_LINE_LOOP

    //draw vertices
    declare sub glVertex2f lib GL_LIB (x as Single, y as Single)
    for i as Single = TWO_PI downto -0.0001 step stp
        glVertex2f centreX + sin(i)*radiusX, centreY + cos(i)*radiusY
    next

else

    //inner radius
    dim innerRadiusX as Single = radiusX - private_penwidth
    dim innerRadiusY as Single = radiusY - private_penheight

    //begin quad strip
    declare sub glBegin lib GL_LIB (mode as UInt32)
    glBegin GL_QUAD_STRIP

    //draw vertices
    declare sub glVertex2f lib GL_LIB (x as Single, y as Single)
    for i as Single = TWO_PI downto -0.0001 step stp
        dim sine as Single = sin(i)
        dim cosine as Single = cos(i)
        glVertex2f centreX + sine*radiusX, centreY + cosine*radiusY
        glVertex2f centreX + sine*innerRadiusX, centreY + cosine*innerRadiusY
    next

end

//end shape
declare sub glEnd lib GL_LIB ()
glEnd
End Sub

```

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(Image As Picture, X As Integer, Y As Integer, Width As Integer, Height As Integer, Orientation as Double, Blend as Boolean)

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//call other drawing method
```

```
DrawPicture Image, X, Y, Width, Height, 0, 0, Image.Width, Image.Height, Orientation, Blend
```

End Sub

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(Image As Picture, X As Integer, Y As Integer, Orientation As Double, Blend as Boolean)

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//call other drawing method
```

```
DrawPicture Image, X, Y, Image.Width, Image.Height, 0, 0, Image.Width, Image.Height, Orientation, Blend
```

End Sub

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(image as Picture, x As Integer, y As Integer, w1 As Integer, h1 As Integer, sx As Integer, sy As Integer, w2 As Integer, h2 As Integer, Orientation as Double, Blend as Boolean)

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//get coordinates
```

```
if w1 = -10000 then w1 = image.Width
```

```
if h1 = -10000 then h1 = image.Height
```

```
if w2 = -10000 then w2 = w1
```

```
if h2 = -10000 then h2 = h1
```

```
//get minimum width and height
```

```
dim minw as integer = min(w1,w2)
```

```
dim minh as integer = min(h1,h2)
```

```
dim tex as RBGLPicture
```

```
dim cached as RBGLPicture = private_cache.Lookup(image,nil)
```

```
if cached <> nil then
```

```
    //get texture from cache
```

```
    tex = cached
```

```
elseif CachePictures = false or (private_cache_size > -1 and private_cache.Count >= private_cache_size) then
```

```
    //crop and resize image prior to building texture
```

```
    if minw < image.Width or minh < image.Height then
```

```

dim temp as Picture
dim p as New Picture(minw,minh,32)
if private_has_mask(image) then

    //copy and resize mask image
    p.mask.graphics.drawPicture image.mask, 0, 0, minw, minh, sx, sy, w2, h2

    //preserve mask image
    temp = NewPicture(w2,h2,32)
    temp.graphics.drawpicture image.mask, 0, 0, w2, h2, sx, sy, w2, h2

    //remove image mask
    image.mask.graphics.forecolor = &c000000
    image.mask.graphics.fillrect sx,sy,w2,h2

end

//copy and resize image
p.graphics.drawPicture image,0,0,minw,minh,sx,sy,w2,h2
p.Transparent = image.Transparent
image = p

//restore mask image
if temp <> nil then image.mask.graphics.drawpicture temp, sx, sy, w2, h2, 0, 0, w2, h2

//update source coordinates
sx = 0
sy = 0
w2 = minw
h2 = minh

end

//convert to texture (don't create mip-maps as image will not be resized)
tex = new RBGLPicture(image)
tex.WrapX = RBGLWrapMode.Clamp
tex.WrapY = RBGLWrapMode.Clamp

else

    //convert to texture
    tex = new RBGLPicture(image)
    tex.WrapX = RBGLWrapMode.Clamp
    tex.WrapY = RBGLWrapMode.Clamp

    //save in cache
    private_cache.Value(image) = tex

end

//draw the image as a texture
DrawPicture tex, x, y, w1, h1, sx, sy, w2, h2, Orientation, Blend
End Sub

```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, Orientation As Double, Blend as Boolean = false)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    if not Blend then
        //set colour to white
        declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
        static white as RBGLColor = RGBA(&cFFFFFF)
        glColor3ubv white
    end

    //draw texture image
    Image.Draw X, Y, Orientation

    if not Blend then
        //restore colour
        declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
        glColor4ubv private_color
    end
End Sub
```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, DestWidth As Single, DestHeight As Single,
SourceX As Single, SourceY As Single, SourceWidth As Single, SourceHeight As Single, Blend as Boolean = false)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //call other drawing method
    DrawPicture Image, x, y, DestWidth, DestHeight, SourceX, SourceY, SourceWidth, SourceHeight, 0.0, Blend
End Sub
```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, DestWidth As Single, DestHeight As Single,
SourceX As Single, SourceY As Single, SourceWidth As Single, SourceHeight As Single, Orientation as Double, Blend
as Boolean = false)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    if not Blend then
        //set colour to white
        declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
```

```

    static white as RBGLColor = RGBA(&cFFFFFF)
    glColor3ubv white
end

```

```

//draw texture image
Image.Draw X, Y, DestWidth, DestHeight, SourceX, SourceY, SourceWidth, SourceHeight, Orientation

```

```

if not Blend then
    //restore colour
    declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
    glColor4ubv private_color
end
End Sub

```

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(Image As Picture, X As Integer, Y As Integer, Width As Integer, Height As Integer, Blend as Boolean)

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

    //call other drawing method
    DrawPicture Image, X, Y, Width, Height, 0, 0, Image.Width, Image.Height, 0.0, Blend

```

End Sub

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(image as Picture, x As Integer, y As Integer, w1 As Integer = - 10000, h1 As Integer = - 10000, sx As Integer = 0, sy As Integer = 0, w2 As Integer = - 10000, h2 As Integer = - 10000)

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

    //call other drawing method
    DrawPicture image, x, y, w1, h1, sx, sy, w2, h2, false

```

End Sub

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(Image As Picture, X As Integer, Y As Integer, Blend as Boolean)

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

    //call other drawing method
    DrawPicture Image, X, Y, Image.Width, Image.Height, 0, 0, Image.Width, Image.Height, 0.0, Blend

```

End Sub

### **RBGLGraphics.DrawPicture:**

Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, Width As Single, Height As Single, Blend as Boolean = false)

```

    #pragma BackgroundTasks false

```

```
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//call other drawing method
DrawPicture Image, x, y, Width, Height, 0.0, Blend
End Sub
```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, Blend as Boolean = false)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //call other drawing method
    DrawPicture Image, x, y, 0.0, Blend
End Sub
```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(image as Picture, x As Integer, y As Integer, w1 As Integer, h1 As Integer, sx As Integer, sy As Integer, w2 As Integer, h2 As Integer, Blend as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //call other drawing method
    DrawPicture Image, x, y, w1, h1, sx, sy, w2, h2, 0.0, Blend
End Sub
```

### **RBGLGraphics.DrawPicture:**

```
Sub DrawPicture(Image As RBGLPicture, X As Single, Y As Single, Width As Single, Height As Single, Orientation As Double, Blend as Boolean = false)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    if not Blend then
        //set colour to white
        declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
        static white as RBGLColor = RGBA(&cFFFFFF)
        glColor3ubv white
    end

    //draw texture image
    Image.Draw X, Y, Width, Height, Orientation

    if not Blend then
        //restore colour
    end
```

```

        declare sub glColor4ubv lib GL_LIB (byref v as RGBColor)
        glColor4ubv private_color
    end
End Sub

```

## **RBGLGraphics.DrawPolygon:**

```

Sub DrawPolygon(Points() As Integer, ZeroBased As Boolean = False)

```

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

    //bind me
    Bind

```

```

    //clear texture
    private_context.Texture = nil

```

```

    //starting index
    dim start as Integer = 1
    if ZeroBased then start = 0

```

```

    if private_pixel_pen then

```

```

        //begin line loop
        declare sub glBegin lib GL_LIB (mode as Uint32)
        glBegin GL_LINE_LOOP

```

```

        //draw vertices
        declare sub glVertex2i lib GL_LIB (x as Integer, y as Integer)
        for i as Integer = start to UBound(Points) step 2
            glVertex2i Points(i), Points(i+1)
        next

```

```

        //end line loop
        declare sub glEnd lib GL_LIB ()
        glEnd

```

```

    else

```

```

        dim x1 as Integer = Points(Points.Ubound-1)
        dim y1 as Integer = Points(Points.Ubound)
        for i as Single = start to UBound(Points) step 2

```

```

            dim x2 as Integer = Points(i)
            dim y2 as Integer = Points(i+1)

```

```

            //line
            DrawLine x1,y1,x2,y2

```

```

            x1 = x2
            y1 = y2

```

```

        next

```



```
end
End Sub
```

### **RBGLGraphics.DrawPolygon:**

```
Sub DrawPolygon(Points() As Double, ZeroBased As Boolean = False)
```

```
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
    //bind me
    Bind
```

```
    //clear texture
    private_context.Texture = nil
```

```
    //starting index
    dim start as Integer = 1
    if ZeroBased then start = 0
```

```
    if private_pixel_pen then
```

```
        //begin line loop
        declare sub glBegin lib GL_LIB (mode as UInt32)
        glBegin GL_LINE_LOOP
```

```
        //draw vertices
        declare sub glVertex2d lib GL_LIB (x as Double, y as Double)
        for i as Integer = start to UBound(Points) step 2
            glVertex2d Points(i) - 0.5, Points(i+1) - 0.5
        next
```

```
        //end line loop
        declare sub glEnd lib GL_LIB ()
        glEnd
```

```
    else
```

```
        dim x1 as Integer = Points(Points.Ubound-1)
        dim y1 as Integer = Points(Points.Ubound)
        for i as Single = start to UBound(Points) step 2
```

```
            dim x2 as Integer = Points(i)
            dim y2 as Integer = Points(i+1)
```

```
            //line
            DrawLine x1,y1,x2,y2
```

```
            x1 = x2
            y1 = y2
```

```
        next
```

```
end
End Sub
```

### **RBGLGraphics.DrawPolygon:**

```
Sub DrawPolygon(Points() As Single, ZeroBased As Boolean = False)
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
//bind me
```

```
Bind
```

```
//clear texture
```

```
private_context.Texture = nil
```

```
//starting index
```

```
dim start as Integer = 1
```

```
if ZeroBased then start = 0
```

```
if private_pixel_pen then
```

```
    //begin line loop
```

```
    declare sub glBegin lib GL_LIB (mode as UInt32)
```

```
    glBegin GL_LINE_LOOP
```

```
    //draw vertices
```

```
    declare sub glVertex2f lib GL_LIB (x as Single, y as Single)
```

```
    for i as Integer = start to UBound(Points) step 2
```

```
        glVertex2f Points(i) - 0.5, Points(i+1) - 0.5
```

```
    next
```

```
    //end line loop
```

```
    declare sub glEnd lib GL_LIB ()
```

```
    glEnd
```

```
else
```

```
    dim x1 as Integer = Points(Points.Ubound-1)
```

```
    dim y1 as Integer = Points(Points.Ubound)
```

```
    for i as Single = start to UBound(Points) step 2
```

```
        dim x2 as Integer = Points(i)
```

```
        dim y2 as Integer = Points(i+1)
```

```
        //line
```

```
        DrawLine x1,y1,x2,y2
```

```
        x1 = x2
```

```
        y1 = y2
```

```
    next
```

```
end
```

End Sub

### **RBGLGraphics.DrawRect:**

Sub DrawRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

//bind me

Bind

//clear texture

private\_context.Texture = nil

//begin quad strip

declare sub glBegin lib GL\_LIB (mode as UInt32)

glBegin GL\_QUAD\_STRIP

//add width and penwidth to dimensions

dim xPlusWidth as Integer = x + Width

dim yPlusHeight as Integer = y + Height

dim xPlusWidthMinusPenWidth as Integer = xPlusWidth - private\_penwidth

dim yPlusHeightMinusPenHeight as Integer = yPlusHeight - private\_penheight

dim xPlusPenWidth as Integer = x + private\_penwidth

dim yPlusPenHeight as Integer = y + private\_penheight

//draw vertices

declare sub glVertex2i lib GL\_LIB (x as Integer, y as Integer)

glVertex2i x, y

glVertex2i xPlusPenWidth, yPlusPenHeight

glVertex2i xPlusWidth, y

glVertex2i xPlusWidthMinusPenWidth, yPlusPenHeight

glVertex2i xPlusWidth, yPlusHeight

glVertex2i xPlusWidthMinusPenWidth, yPlusHeightMinusPenHeight

glVertex2i x, yPlusHeight

glVertex2i xPlusPenWidth, yPlusHeightMinusPenHeight

glVertex2i x, y

glVertex2i xPlusPenWidth, yPlusPenHeight

//end quad strip

declare sub glEnd lib GL\_LIB ()

glEnd

End Sub

### **RBGLGraphics.DrawRoundRect:**

Sub DrawRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight As Integer)

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

```

//bind me
Bind

//clear texture
private_context.Texture = nil

//begin polygon
declare sub glBegin lib GL_LIB (mode as UInt32)
glBegin GL_QUAD_STRIP

//steps
dim radius as Single = (width+height)/2
dim circumference as Single = TWO_PI*radius
dim steps as Integer = max(MIN_CIRCLE_EDGES,circumference/MAX_CIRCLE_EDGE_LENGTH)/4
dim stp as Single = HALF_PI/steps

//radius
dim radiusX as Single = ArcWidth/2
dim innerRadiusX as Single = max(radiusX,PenWidth) - PenWidth
dim radiusY as Single = ArcHeight/2
dim innerRadiusY as Single = max(radiusY,PenHeight) - PenHeight

//draw vertices
declare sub glVertex2f lib GL_LIB (x as Single, y as Single)

//top left corner
dim centreX as Single = x + radiusX
dim centreY as Single = y + radiusY
dim innerCentreX as Single = x + max(radiusX,PenWidth)
dim innerCentreY as Single = y + max(radiusY,PenHeight)
for i as Single = HALF_PI downto -0.0001 step stp
    dim sine as Single = sin(i)
    dim cosine as Single = cos(i)
    glVertex2f centreX - sine*radiusX, centreY - cosine*radiusY
    glVertex2f innerCentreX - sine*innerRadiusX, innerCentreY - cosine*innerRadiusY
next

//top right corner
centreX = x + width - radiusX
innerCentreX = x + width - max(radiusX,PenWidth)
for i as Single = HALF_PI downto -0.0001 step stp
    dim sine as Single = sin(i)
    dim cosine as Single = cos(i)
    glVertex2f centreX + cosine*radiusX, centreY - sine*radiusY
    glVertex2f innerCentreX + cosine*innerRadiusX, innerCentreY - sine*innerRadiusY
next

//bottom right corner
centreY = y + height - radiusY
innerCentreY = y + height - max(radiusY,PenHeight)
for i as Single = HALF_PI downto -0.0001 step stp
    dim sine as Single = sin(i)
    dim cosine as Single = cos(i)
    glVertex2f centreX + sine*radiusX, centreY + cosine*radiusY

```

```

    glVertex2f innerCentreX + sine*innerRadiusX, innerCentreY + cosine*innerRadiusY
next

//bottom left corner
centreX = x + radiusX
innerCentreX = x + max(radiusX, PenWidth)
for i as Single = HALF_PI downto -0.0001 step stp
    dim sine as Single = sin(i)
    dim cosine as Single = cos(i)
    glVertex2f centreX - cosine*radiusX, centreY + sine*radiusY
    glVertex2f innerCentreX - cosine*innerRadiusX, innerCentreY + sine*innerRadiusY
next

//last two points
glVertex2f x, y + radiusY
glVertex2f x + PenWidth, y + max(radiusY, PenHeight)

//end polygon
declare sub glEnd lib GL_LIB ()
glEnd
End Sub

```

### **RBGLGraphics.DrawStopIcon:**

```

Sub DrawStopIcon(X As Integer, Y As Integer, Blend as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //build texture
    private_build_icon_texture STOP_ICON

    //draw icon
    DrawPicture private_icons(STOP_ICON), X, Y, Blend
End Sub

```

### **RBGLGraphics.DrawStopIcon:**

```

Sub DrawStopIcon(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //call other drawstopicon method
    DrawStopIcon X, Y, False
End Sub

```

### **RBGLGraphics.DrawString:**

```

Sub DrawString(Text As String, X As Integer, Y As Integer, WrapWidth As Integer, Condense As Boolean, Blend As Boolean)

```

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//bind me
Bind

//update string properties
private_string.Text = Text
private_string.WrapWidth = WrapWidth
private_string.Condense = Condense

//check cache
dim s as RBGLStringTexture
dim key as String = private_string_cache_key(text)
dim value as Variant = private_cache.Lookup(key,nil)
dim cached as RBGLStringTexture
if value <> nil then

    //check that match is exact (hashes are not unique)
    #if RBVersion < 2008.01 then
        //use the soon-to-be-deprecated collection class
        dim vals as Collection = value
        for i as integer = 1 to vals.count step 2
            if StrComp(vals.item(i).StringValue, text, 0) = 0 then
                cached = vals.item(i+1)
                exit
            end
        next
    #else
        //use an array as the dictionary value
        dim vals() as Variant = value
        for i as integer = 0 to vals.Ubound step 2
            if StrComp(vals(i).StringValue, text, 0) = 0 then
                cached = vals(i+1)
                exit
            end
        next
    #endif

end

if cached <> nil then

    //use cached string
    s = cached

elseif CacheStrings = false or (private_cache_size > -1 and private_cache.Count >= private_cache_size) then

    //use common string
    s = private_string

else

```

```

//create new string
s = new RBGLStringTexture(text)
s.WrapWidth = WrapWidth
s.Condense = Condense
s.Bold = bold
s.Italic = Italic
s.Underline = Underline
s.FontName = TextFont
s.FontSize = TextSize

//add to cache
if value = nil then
    //new entry
    #if RBVersion < 2008.01 then
        dim c as new Collection
        c.add text
        c.add s
        private_cache.Value(key) = c
    #else
        private_cache.Value(key) = array(text, s)
    #endif
else
    //append to existing entry
    #if RBVersion < 2008.01 then
        dim c as Collection = value
        c.add text
        c.add s
    #else
        dim vals() as Variant = value
        vals.Append text
        vals.Append s
    #endif
end

end

if Blend then

    //draw string as a texture
    s.Draw X, Y

else

    static black as RBGLColor = RGBA(&c000000)

    //set colour to black
    declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
    glColor3ubv black

    //draw string as a texture
    s.Draw X, Y

    //restore colour

```

```
declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
glColor4ubv private_color
```

```
end
End Sub
```

### **RBGLGraphics.DrawString:**

```
Sub DrawString(Text As String, X As Integer, Y As Integer, WrapWidth As Integer = 0, Condense As Boolean = False)
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//call other string method
DrawString Text, X, Y, WrapWidth, Condense, True
End Sub
```

### **RBGLGraphics.FillOval:**

```
Sub FillOval(X As Integer, Y As Integer, Width As Integer, Height As Integer)
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//bind me
Bind

//clear texture
private_context.Texture = nil

//begin polygon
declare sub glBegin lib GL_LIB (mode as UInt32)
glBegin GL_POLYGON

//steps
dim radius as Single = (width+height)/2
dim circumference as Single = TWO_PI*radius
dim steps as Integer = max(MIN_CIRCLE_EDGES,circumference/MAX_CIRCLE_EDGE_LENGTH)
dim stp as Single = TWO_PI/steps

//centre and radius
dim radiusX as Single = width/2
dim centreX as Single = x + radiusX
dim radiusY as Single = height/2
dim centreY as Single = y + radiusY

//draw vertices
declare sub glVertex2f lib GL_LIB (x as Single, y as Single)
for i as Single = TWO_PI downto 0.0001 step stp
    glVertex2f centreX + sin(i)*radiusX, centreY + cos(i)*radiusY
next

//end polygon
declare sub glEnd lib GL_LIB ()
```



```
glEnd
End Sub
```

### **RBGLGraphics.FillPolygon:**

```
Sub FillPolygon(Points() As Double, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //clear texture
    private_context.Texture = nil

    //begin polygon
    declare sub glBegin lib GL_LIB (mode as UInt32)
    glBegin GL_POLYGON

    //starting index
    dim start as Integer = 1
    if ZeroBased then start = 0

    //draw vertices
    declare sub glVertex2d lib GL_LIB (x as Double, y as Double)
    for i as Integer = start to UBound(Points) step 2
        glVertex2d Points(i), Points(i+1)
    next

    //end polygon
    declare sub glEnd lib GL_LIB ()
    glEnd
End Sub
```

### **RBGLGraphics.FillPolygon:**

```
Sub FillPolygon(Points() As Single, ZeroBased As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    //clear texture
    private_context.Texture = nil

    //begin polygon
    declare sub glBegin lib GL_LIB (mode as UInt32)
    glBegin GL_POLYGON

    //starting index
    dim start as Integer = 1
```

```
if ZeroBased then start = 0
```

```
//draw vertices  
declare sub glVertex2f lib GL_LIB (x as Single, y as Single)  
for i as Integer = start to UBound(Points) step 2  
    glVertex2f Points(i), Points(i+1)  
next
```

```
//end polygon  
declare sub glEnd lib GL_LIB ()  
glEnd
```

```
End Sub
```

### **RBGLGraphics.FillPolygon:**

```
Sub FillPolygon(Points() As Integer, ZeroBased As Boolean = False)
```

```
#pragma BackgroundTasks false  
#pragma BoundsChecking DebugBuild  
#pragma NilObjectChecking DebugBuild  
#pragma StackOverflowChecking false
```

```
//bind me  
Bind
```

```
//clear texture  
private_context.Texture = nil
```

```
//begin polygon  
declare sub glBegin lib GL_LIB (mode as UInt32)  
glBegin GL_POLYGON
```

```
//starting index  
dim start as Integer = 1  
if ZeroBased then start = 0
```

```
//draw vertices  
declare sub glVertex2i lib GL_LIB (x as Integer, y as Integer)  
for i as Integer = start to UBound(Points) step 2  
    glVertex2i Points(i), Points(i+1)  
next
```

```
//end polygon  
declare sub glEnd lib GL_LIB ()  
glEnd
```

```
End Sub
```

### **RBGLGraphics.FillRect:**

```
Sub FillRect(X As Integer, Y As Integer, Width As Integer, Height As Integer)
```

```
#pragma BackgroundTasks false  
#pragma BoundsChecking DebugBuild  
#pragma NilObjectChecking DebugBuild  
#pragma StackOverflowChecking false
```

```
//bind me  
Bind
```

```

//clear texture
private_context.Texture = nil

//draw rect
declare sub glRecti lib GL_LIB (x1 as Integer, y1 as Integer, x2 as Integer, y2 as Integer)
glRecti x, y, x + Width, y + Height
End Sub

```

### **RBGLGraphics.FillRoundRect:**

```

Sub FillRoundRect(X As Integer, Y As Integer, Width As Integer, Height As Integer, ArcWidth As Integer, ArcHeight
As Integer)

```

```

    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

```

```

//bind me
Bind

```

```

//clear texture
private_context.Texture = nil

```

```

//begin polygon
declare sub glBegin lib GL_LIB (mode as UInt32)
glBegin GL_POLYGON

```

```

//steps
dim radius as Single = (width+height)/2
dim circumference as Single = TWO_PI*radius
dim steps as Integer = max(MIN_CIRCLE_EDGES, circumference/MAX_CIRCLE_EDGE_LENGTH)/4
dim stp as Single = HALF_PI/steps

```

```

//radius
dim radiusX as Single = ArcWidth/2
dim radiusY as Single = ArcHeight/2

```

```

//draw vertices
declare sub glVertex2f lib GL_LIB (x as Single, y as Single)

```

```

//top left corner
dim centreX as Single = x + radiusX
dim centreY as Single = y + radiusY
for i as Single = HALF_PI downto -0.0001 step stp
    glVertex2f centreX - sin(i)*radiusX, centreY - cos(i)*radiusY
next

```

```

//top right corner
centreX = x + width - radiusX
for i as Single = HALF_PI downto -0.0001 step stp
    glVertex2f centreX + cos(i)*radiusX, centreY - sin(i)*radiusY
next

```

```

//bottom right corner

```

```

centreY = y + height - radiusY
for i as Single = HALF_PI downto -0.0001 step stp
    glVertex2f centreX + sin(i)*radiusX, centreY + cos(i)*radiusY
next

//bottom left corner
centreX = x + radiusX
for i as Single = HALF_PI downto -0.0001 step stp
    glVertex2f centreX - cos(i)*radiusX, centreY + sin(i)*radiusY
next

//end polygon
declare sub glEnd lib GL_LIB ()
    glEnd
End Sub

```

### **RBGLGraphics.Flush:**

```

Sub Flush()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //flush context
    private_context.Flush
End Sub

```

### **RBGLGraphics.ForeColor:**

```

Sub ForeColor(Assigns Col As Color)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set colour
    private_color = RGBA(Col)

    //apply colour
    declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
    glColor3ubv private_color
End Sub

```

### **RBGLGraphics.ForeColor:**

```

Sub ForeColor(Assigns Color As UInt32)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set colour
    private_color = RGBA(Color)

    //apply colour

```

```

declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
glColor4ubv private_color
End Sub

```

### **RBGLGraphics.ForeColor:**

```

Function ForeColor() As Color
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_color.RGBColor
End Function

```

### **RBGLGraphics.GetPicture:**

```

Function GetPicture(X As Integer = 0, Y As Integer = 0, Width As Integer = -10000, Height As Integer = -10000)
As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get realbasic picture
    return GetRBGLPicture(X, Y, Width, Height).GetPicture

End Function

```

### **RBGLGraphics.GetRBGLPicture:**

```

Function GetRBGLPicture(X As Integer = 0, Y As Integer = 0, Width As Integer = -10000, Height As Integer =
-10000) As RBGLPicture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //transform coordinates
    if Width = -10000 then width = private_rect.Width
    if Height = -10000 then height = private_rect.Height
    X = X + private_rect.Left
    Y = Y + private_rect.Top

    //get rbglpicture
    return Context.GetRBGLPicture(X, Y, Width, Height)
End Function

```

### **RBGLGraphics.Handle:**

```

Function Handle() As UInt32
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_context = nil then

```

```

        return 0
    else
        return private_context.Handle
    end
End Function

```

### **RBGLGraphics.Height:**

```

Function Height() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_rect.Height
End Function

```

### **RBGLGraphics.Italic:**

```

Function Italic() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.Italic

End Function

```

### **RBGLGraphics.Italic:**

```

Sub Italic(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_string.Italic = value

End Sub

```

### **RBGLGraphics.PenHeight:**

```

Sub PenHeight(assigns value as Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_penheight = value

End Sub

```

### **RBGLGraphics.PenHeight:**

```

Function PenHeight() As Integer

```

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_penheight
```

End Function

### **RBGLGraphics.PenWidth:**

Sub PenWidth(assigns value as Integer)

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
private_penwidth = value
```

End Sub

### **RBGLGraphics.PenWidth:**

Function PenWidth() As Integer

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_penwidth
```

End Function

### **RBGLGraphics.Pixel:**

Function Pixel(X As Integer, Y As Integer) As Color

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//bind me
```

```
Bind
```

```
//get pixel
```

```
dim col as RBGLColor
```

```
declare sub glReadPixels lib GL_LIB (x as integer, y as integer, _
width as integer, height as integer, format as uint32, type as uint32, byref col as RBGLColor)
```

```
glReadPixels private_rect.Left + x,private_context.Height - private_rect.Top - y - 1, _
1,1,GL_RGBA,GL_UNSIGNED_BYTE,col
```

```
return col.RGBColor
```

End Function

### **RBGLGraphics.Pixel:**

Sub Pixel(X As Integer, Y As Integer, Assigns Col As RBGLColor)

```
#pragma BackgroundTasks false
```

```

#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//bind me
Bind

//clear texture
private_context.Texture = nil

//set cursor position
declare sub glRasterPos2i lib GL_LIB (x as integer, y as integer)
glRasterPos2i x,y+1

//draw pixel
declare sub glDrawPixels lib GL_LIB (width as integer, height as integer, format as uint32, type as uint32, byref
col as RBGLColor)
glDrawPixels 1,1,GL_RGBA,GL_UNSIGNED_BYTE,Col
End Sub

```

### **RBGLGraphics.Pixel:**

```

Sub Pixel(X As Integer, Y As Integer, Assigns Col As UInt32)
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

Pixel(x,y) = RGBA(Col)
End Sub

```

### **RBGLGraphics.Pixel:**

```

Sub Pixel(X As Integer, Y As Integer, Assigns Col As Color)
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

Pixel(x,y) = RGBA(Col)
End Sub

```

### **RBGLGraphics.private\_build\_icon\_texture:**

```

Private Shared Sub private_build_icon_texture(IconIndex as Integer)
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//exit if already built
if private_icons(IconIndex) <> nil then
return
end

//black background

```



```

dim pBlack as new Picture(32,32,32)
dim bs as RGBSurface = pBlack.RGBSurface
pBlack.Graphics.ForeColor = &c000000
pBlack.Graphics.FillRect 0,0,32,32

//white background
dim pWhite as new Picture(32,32,32)
dim ws as RGBSurface = pWhite.RGBSurface

//draw icon
select case IconIndex
case CAUTION_ICON
    pBlack.Graphics.DrawCautionIcon 0,0
    pWhite.Graphics.DrawCautionIcon 0,0
case NOTE_ICON
    pBlack.Graphics.DrawNotelIcon 0,0
    pWhite.Graphics.DrawNotelIcon 0,0
case STOP_ICON
    pBlack.Graphics.DrawStopIcon 0,0
    pWhite.Graphics.DrawStopIcon 0,0
end

//compare pixels and calculate mask value
dim ms as RGBSurface = pBlack.mask.RGBSurface
for j as integer = 0 to 31
    for k as integer = 0 to 31
        dim cb as Color = bs.Pixel(k,j)
        dim cw as Color = ws.Pixel(k,j)
        dim o as integer = max(cw.red - cb.red,max(cw.green - cb.green,cw.blue - cb.blue))
        ms.Pixel(k,j) = rgb(o,o,o)
    next
next

//create icon texture (don't use mip-mapping as icon size is fixed)
private_icons(IconIndex) = new RBGLPicture(pBlack, RBGLPixelFormat.RGBA)
End Sub

```

### **RBGLGraphics.private\_string\_cache\_key:**

Private Function private\_string\_cache\_key(text as String) As String

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

```

```

dim s as String = str(private_string.WrapWidth)
if private_string.condense then s = s + "c"
if private_string.Bold then s = s + "b"
if private_string.Italic then s = s + "i"
if private_string.Underline then s = s + "u"
s = s + str(private_string.FontSize)
s = s + (private_string.FontName + "|")

```

```

dim v as Variant = text
return s + Str(v.Hash) //short but may not be unique

```

End Function

### **RBGLGraphics.RGBAPixel:**

Function RGBAPixel(X As Integer, Y As Integer) As RBGLColor

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

//bind me

Bind

//get pixel

dim col as RBGLColor

declare sub glReadPixels lib GL\_LIB (x as integer, y as integer, \_

width as integer, height as integer, format as uint32, type as uint32, byref col as RBGLColor)

glReadPixels private\_rect.Left + x,private\_context.Height - private\_rect.Top - y - 1, \_

1,1,GL\_RGBA,GL\_UNSIGNED\_BYTE,col

return col

End Function

### **RBGLGraphics.StringDirection:**

Function StringDirection(Text As String) As Integer

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

private\_string.Text = Text

return private\_string.TextDirection

End Function

### **RBGLGraphics.StringHeight:**

Function StringHeight(Text As String, WrapWidth As Integer) As Integer

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

private\_string.Text = Text

private\_string.WrapWidth = WrapWidth

return private\_string.Height

End Function

### **RBGLGraphics.StringWidth:**

Function StringWidth(Text As String) As Double

#pragma BackgroundTasks false

#pragma BoundsChecking DebugBuild

#pragma NilObjectChecking DebugBuild

#pragma StackOverflowChecking false

private\_string.Text = Text

return private\_string.Width

End Function

### **RBGLGraphics.TextAscent:**

```
Function TextAscent() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.Ascent
End Function
```

### **RBGLGraphics.TextFont:**

```
Sub TextFont(assigns value as String)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_string.FontName = value

End Sub
```

### **RBGLGraphics.TextFont:**

```
Function TextFont() As String
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.FontName

End Function
```

### **RBGLGraphics.TextHeight:**

```
Function TextHeight() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.LineHeight
End Function
```

### **RBGLGraphics.TextSize:**

```
Function TextSize() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.FontSize
End Function
```

## **RBGLGraphics.TilePicture:**

Sub TilePicture(Image As Picture, X As Single, Y As Single, Width As Single, Height As Single, OffsetX As Single = 0, OffsetY As Single = 0, Blend As Boolean = False)

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
//bind me
```

```
Bind
```

```
dim tex as RBGLPicture
```

```
dim cached as RBGLPicture = private_cache.Lookup(image,nil)
```

```
if cached <> nil then
```

```
    //get texture from cache
```

```
    tex = cached
```

```
elseif CachePictures = false or (private_cache_size > -1 and private_cache.Count >= private_cache_size) then
```

```
    //crop and resize image prior to building texture
```

```
    if width < image.Width or height < image.Height then
```

```
        dim temp as Picture
```

```
        dim p as New Picture(width,height,32)
```

```
        if private_has_mask(image) then
```

```
            //copy and resize mask image
```

```
            p.mask.graphics.drawPicture image.mask,0,0,width,height
```

```
            //preserve mask image
```

```
            temp = NewPicture(width,height,32)
```

```
            temp.graphics.drawpicture image.mask,0,0
```

```
            //remove image mask
```

```
            image.mask.graphics.forecolor = &c000000
```

```
            image.mask.graphics.fillrect 0,0,width,height
```

```
        end
```

```
        //copy and resize image
```

```
        p.graphics.drawPicture image,0,0,width,height
```

```
        p.Transparent = image.Transparent
```

```
        image = p
```

```
        //restore mask image
```

```
        if temp <> nil then image.mask.graphics.drawpicture temp,0,0
```

```
        //convert to texture
```

```
        tex = new RBGLPicture(image)
```

```
        tex.WrapX = RBGLWrapMode.Clamp
```

```
        tex.WrapY = RBGLWrapMode.Clamp
```

```

end

else

    //convert to texture
    tex = new RBGLPicture(image)
    tex.WrapX = RBGLWrapMode.Clamp
    tex.WrapY = RBGLWrapMode.Clamp

    //save in cache
    private_cache.Value(image) = tex

end

//draw the image as a texture
TilePicture tex, X, Y, Width, Height, OffsetX, OffsetY, Blend
End Sub

```

### **RBGLGraphics.TilePicture:**

```

Sub TilePicture(Image As RBGLPicture, X As Single, Y As Single, Width As Single, Height As Single, OffsetX As Single
= 0, OffsetY As Single = 0, Blend As Boolean = False)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //bind me
    Bind

    if not Blend then
        //set colour to white
        declare sub glColor3ubv lib GL_LIB (byref v as RBGLColor)
        static white as RBGLColor = RGBA(&cFFFFFF)
        glColor3ubv white
    end

    //draw texture image
    Image.Tile X, Y, Width, Height, OffsetX, OffsetY

    if not Blend then
        //restore colour
        declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
        glColor4ubv private_color
    end
End Sub

```

### **RBGLGraphics.UnCache:**

```

Shared Sub UnCache(Image As Picture)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_cache <> nil and private_cache.HasKey(Image) then

```

```
        private_cache.Remove Image
    end
End Sub
```

### **RBGLGraphics.Underline:**

```
Sub Underline(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_string.Underline = value

End Sub
```

### **RBGLGraphics.Underline:**

```
Function Underline() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.Underline

End Function
```

### **RBGLGraphics.UseOldRenderer:**

```
Sub UseOldRenderer(assigns value as Boolean)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //does nothing
    private_useoldrenderer = value
End Sub
```

### **RBGLGraphics.UseOldRenderer:**

```
Function UseOldRenderer() As Boolean
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //does nothing
    return private_useoldrenderer

End Function
```

### **RBGLGraphics.Width:**

```
Function Width() As Integer
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_rect.Width
```

```
End Function
```

## **RBGLGraphics.Bold:**

Bold As Boolean

```
Get
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_string.Bold
```

```
End Get
```

```
Set
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
private_string.Bold = value
```

```
End Set
```

```
End Property
```

Shared CachePictures As Boolean

## **RBGLGraphics.CacheSize:**

Shared CacheSize As Integer

```
Get
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
return private_cache_size
```

```
End Get
```

```
Set
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
//set cache size
```

```
private_cache_size = value
```

```
//clear cache
```

```
if private_cache = nil then
```

```
    private_cache = new Dictionary
```

```
elseif private_cache_size > -1 and private_cache_size < private_cache.Count then
```

```
    private_cache.Clear
```

```
end
End Set
End Property
Shared CacheStrings As Boolean
```

### **RBGLGraphics.Context:**

Context As RBGLContext

```
Get
    return private_context
End Get
End Property
```

### **RBGLGraphics.ForeColor:**

ForeColor As RBGLColor

```
Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_color
End Get
Set
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set colour
    private_color = value

    //apply colour
    declare sub glColor4ubv lib GL_LIB (byref v as RBGLColor)
    glColor4ubv private_color
End Set
End Property
```

### **RBGLGraphics.Height:**

Height As Integer

```
Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_rect.Height
End Get
End Property
```



## **RBGLGraphics.Italic:**

Italic As Boolean

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_string.Italic
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
private_string.Italic = value
```

End Set

End Property

## **RBGLGraphics.PenHeight:**

PenHeight As Single

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_penheight
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
private_penheight = value
```

```
private_pixel_pen = (private_penheight = 1.0 and private_penwidth = 1.0)
```

End Set

End Property

## **RBGLGraphics.PenWidth:**

PenWidth As Single

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_penwidth
```

```

End Get
Set
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_penwidth = value
    private_pixel_pen = (private_penheight = 1.0 and private_penwidth = 1.0)
End Set
End Property
Private Shared private_cache As Dictionary

Private Shared private_cache_size As Integer

Private private_color As RBGLColor

Private private_context As RBGLContext

Private Shared private_icons(2) As RBGLPicture

Private private_penheight As Single

Private private_penwidth As Single

Private private_pixel_pen As boolean

Private private_rect As GLIntRect

Private private_string As RBGLStringTexture

Private private_useoldrenderer As Boolean

```

### **RBGLGraphics.TextAscent:**

TextAscent As Integer

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.Ascent
End Get
End Property

```

### **RBGLGraphics.TextFont:**

TextFont As String

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild

```

```

        #pragma StackOverflowChecking false

        return private_string.FontName
    End Get
    Set
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        private_string.FontName = value
    End Set
End Property

```

### **RBGLGraphics.TextHeight:**

TextHeight As Integer

```

    Get
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        return private_string.LineHeight
    End Get
End Property

```

### **RBGLGraphics.TextSize:**

TextSize As Integer

```

    Get
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        return private_string.FontSize
    End Get
    Set
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        private_string.FontSize = value
    End Set
End Property

```

### **RBGLGraphics.Underline:**

Underline As Boolean

```

    Get
        #pragma BackgroundTasks false

```

```

    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_string.Underline
End Get
Set
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    private_string.Underline = value
End Set
End Property

```

## **RBGLGraphics.Width:**

Width As Integer

```

Get
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    return private_rect.Width
End Get
End Property
End Class
Class RBGLPicture

```

```

Private Const GL_LINEAR = 9729
Private Const GL_LINEAR_MIPMAP_LINEAR = 9987
Private Const GL_NEAREST = &h2600
Private Const GL_NEAREST_MIPMAP_LINEAR = &h2702
Private Const GL_PACK_ALIGNMENT = 3333
Private Const GL_QUADS = 7
Private Const GL_TEXTURE_2D = 3553
Private Const GL_TEXTURE_MAG_FILTER = 10240
Private Const GL_TEXTURE_MIN_FILTER = 10241
Private Const GL_TEXTURE_WRAP_S = 10242
Private Const GL_TEXTURE_WRAP_T = 10243
Private Const GL_UNSIGNED_BYTE = 5121

```

## **RBGLPicture.Bind:**

```

Sub Bind()
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get current context
    if private_current_context = nil or private_current_context.value = nil then return
    dim context as RBGLContext = RBGLContext(private_current_context.value)

```

```

dim id as uint32
if private_ids_by_context.HasKey(context.Handle) then

    //texture has already been created in this context
    id = private_ids_by_context.value(context.Handle)

    //bind texture
    declare sub glBindTexture lib GL_LIB (target as uint32, texture as uint32)
    glBindTexture GL_TEXTURE_2D, id

else

    //need to create texture for this context

    //generate id
    declare sub glGenTextures lib GL_LIB (count as uint32, byref tex as uint32)
    glGenTextures 1, id

    //bind texture
    declare sub glBindTexture lib GL_LIB (target as uint32, texture as uint32)
    glBindTexture GL_TEXTURE_2D, id

    //create texture
    declare function gluBuild2DMipmaps lib GLU_LIB (target as uint32, internalformat as uint32, width as uint32,
    height as uint32, format as uint32, type as uint32, data as Ptr) as integer
    if not private_mipmapped or gluBuild2DMipmaps(GL_TEXTURE_2D, private_format, private_actual_width,
    private_actual_height, private_format, GL_UNSIGNED_BYTE, private_data) <> 0 then
        private_mipmapped = false
        //set image level zero
        declare sub glTexImage2D lib GL_LIB (target as uint32, level as uint32, internalformat as uint32, width as
        uint32, height as uint32, border as uint32, format as uint32, type as uint32, data as ptr)
        glTexImage2D GL_TEXTURE_2D, 0, private_format, private_actual_width, private_actual_height, 0,
        private_format, GL_UNSIGNED_BYTE, private_data
    end

    //add id to dictionary
    private_ids_by_context.Value(context.Handle) = id

end

//set filtering mode
declare sub glTexParameterI lib GL_LIB (target as uint32, pname as uint32, param as uint32)
if private_mipmapped then
    if private_downscaling = RBGLScalingMode.Linear then
        glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR
    else
        glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR
    end
else
    glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, UInt32(private_downscaling)
end
glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, UInt32(private_upscaling)

```

```

//set x wrap mode
gltexParameteri GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, UInt32(private_wrapx)

//set y wrap mode
gltexParameteri GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, UInt32(private_wrapy)

//set current context's texture to me
//this will cause texture to be bound twice if bind method is
//not called from within the Texture property setter of RBGLContext
context.Texture = me
End Sub

```

### **RBGLPicture.Blit:**

```

Sub Blit(X As Integer, Y As Integer, Width As Single = - 10000, Height As Single = - 10000)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //set default width and height
    if Width = -10000 and Height = -10000 then
        Width = private_width
        Height = private_height
    end

    //set cursor position
    declare sub glRasterPos2i lib GL_LIB (x as Integer, y as Integer)
    glRasterPos2i x,y

    //flip image
    declare sub glPixelZoom lib GL_LIB (xfactor as Single, yfactor as Single)
    glPixelZoom Width/private_actual_width, -Height/private_actual_height

    //draw pixel
    declare sub glDrawPixels lib GL_LIB (width as integer, height as integer, format as uint32, type as uint32, pixels
    as Ptr)
    glDrawPixels private_actual_width, private_actual_height, private_format, GL_UNSIGNED_BYTE, private_data
End Sub

```

### **RBGLPicture.Constructor:**

```

Sub Constructor(Image as Picture, Format as RBGLPixelFormat)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //initialise image
    Constructor image, format, DefaultOptions
End Sub

```

### **RBGLPicture.Constructor:**

```

Sub Constructor(Image as Picture, Format as RBGLPixelFormat, Options as RBGLPictureOptions)
    #pragma BackgroundTasks false

```

```

#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//initialise context id map dictionary
private_ids_by_context = new Dictionary

//get original image size
private_width = image.width
private_height = image.Height

//clean up and resize image if required
image = private_sanitise_picture(image, Options.ScaleToPowerOfTwo)

//get texture dimensions
dim width as uint32 = image.Width
dim height as uint32 = image.Height

//set padded property
private_padded = (not Options.ScaleToPowerOfTwo) and (width > private_width or height > private_height)

#if TargetMacOS

    //use mac-specific functions to copy image data
    private_copy_mac_picture_data image, format

#else

    //use cross-platform realbasic functions to copy image data

    //initialise variables
    dim index as uint32
    dim data as MemoryBlock
    dim widthBound as uint32 = width - 1
    dim heightBound as uint32 = height - 1
    dim c as Color

    //get image rgbsurface
    dim s as RGBSurface = image.RGBSurface
    if s = nil then
        dim p as Picture = newPicture(width,height,32)
        p.graphics.drawPicture image,0,0
        s = p.RGBSurface
    end

    //recode image data to match selected format
    select case format

    case RBGLPixelFormat.RGB

        data = newMemoryBlock(3*width*height)
        data.LittleEndian = false
        for i as uint32 = heightBound downto 0
            for j as uint32 = 0 to widthBound

```

```

        index = (i*width + j)*3
        data.ColorValue(index,24) = s.pixel(j,i)
    next
next

```

case RBGLPixelFormat.RGBA

if private\_has\_mask(image) then

```

//get mask rgbsurface
dim ms as RGBSurface = image.mask.RGBSurface
if ms = nil then
    dim p as Picture = newpicture(width,height,32)
    p.graphics.forecolor = &c000000
    p.graphics.fillrect 0,0,width,height
    ms = p.RGBSurface
end

data = newMemoryBlock(4*width*height)
data.LittleEndian = false
for i as uint32 = heightBound downto 0
    for j as uint32 = 0 to widthBound
        index = (i*width + j)*4
        data.ColorValue(index,24) = s.pixel(j,i)
        data.UInt8Value(index + 3) = 255 - ms.pixel(j,i).red
    next
next

```

elseif image.Transparent = Picture.TransparentWhite then

```

//use the colour white as transparent
data = newMemoryBlock(4*width*height)
data.LittleEndian = false
for i as uint32 = heightBound downto 0
    for j as uint32 = 0 to widthBound
        index = (i*width + j)*4
        c = s.pixel(j,i)
        data.ColorValue(index,24) = c
        if c <> &FFFFFF then data.UInt8Value(index + 3) = 255
    next
next

```

else

```

//mask is opaque
data = newMemoryBlock(4*width*height)
data.LittleEndian = false
for i as uint32 = heightBound downto 0
    for j as uint32 = 0 to widthBound
        index = (i*width + j)*4
        c = s.pixel(j,i)
        data.ColorValue(index,24) = c
        data.UInt8Value(index + 3) = 255
    next

```



```

        next

    end

case RBGLPixelFormat.Luminance

    data = newMemoryBlock(width*height)
    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            data.UInt8Value(i*width + j) = s.pixel(j,i).red
        next
    next

case RBGLPixelFormat.Alpha

    //assumes alpha is stored in main picture, not mask.
    //to extract mask alpha, pass mask as image parameter
    data = newMemoryBlock(width*height)
    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            data.UInt8Value(i*width + j) = 255 - s.pixel(j,i).red
        next
    next

case RBGLPixelFormat.LuminanceAlpha

    //get mask rgbsurface
    dim ms as RGBSurface = image.mask.RGBSurface
    if ms = nil then
        dim p as Picture = newpicture(width,height,32)
        p.graphics.forecolor = &c000000
        p.graphics.fillrect 0,0,width,height
        ms = p.RGBSurface
    end

    data = newMemoryBlock(2*width*height)
    for i as uint32 = heightBound DownTo 0
        for j as uint32 = 0 to widthBound
            index = (i*width + j)*2
            data.UInt8Value(index) = s.pixel(j,i).red
            data.UInt8Value(index + 1) = 255 - ms.pixel(j,i).red
        next
    next

else
    raise New RBGLException("Unsupported image format")
end

//set image attributes
private_data = data
private_format = UInt32(Format)
private_actual_width = width
private_actual_height = height

```

```
#endif
```

```
//set texture attributes  
private_mipmapped = Options.GenerateMipMaps  
private_wrapx = Options.WrapX  
private_wrapy = Options.WrapY  
private_upscaling = Options.Upscaling  
private_downscaling = Options.Downscaling
```

```
End Sub
```

### **RBGLPicture.Constructor:**

```
Sub Constructor(Image as Picture, Options as RBGLPictureOptions)
```

```
#pragma BackgroundTasks false  
#pragma BoundsChecking DebugBuild  
#pragma NilObjectChecking DebugBuild  
#pragma StackOverflowChecking false
```

```
//determine if image has a mask  
dim format as RBGLPixelFormat  
if private_has_mask(image) then
```

```
    format = RBGLPixelFormat.RGBA
```

```
elseif image.Transparent = Picture.TransparentWhite then
```

```
    //clean up and resize image if required  
    image = private_clone_picture(image)
```

```
    //determine if any pixels are transparent  
    dim opaque as Boolean = true  
    dim s as RGBSurface = image.RGBSurface  
    for i as uint32 = image.Height - 1 downto 0  
        for j as uint32 = image.Width - 1 downto 0  
            if s.pixel(j,i) = &cFFFFFF then  
                opaque = false  
                exit  
            end  
        next  
    next
```

```
    if opaque then  
        format = RBGLPixelFormat.RGB  
    else  
        format = RBGLPixelFormat.RGBA  
    end
```

```
else
```

```
    format = RBGLPixelFormat.RGB
```

```
end
```

```
//initialise image
```

```
    Constructor image, format, options
End Sub
```

### **RBGLPicture.Constructor:**

```
Sub Constructor(Image as Picture)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //initialise image
    Constructor image, DefaultOptions
End Sub
```

### **RBGLPicture.Constructor:**

```
Sub Constructor(Context as RBGLContext, X As Integer, Y As Integer, Width As Integer, Height As Integer, Options
As RBGLPictureOptions)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //initilise context id map dictionarys
    private_ids_by_context = new Dictionary

    //bind context
    context.Bind

    //set format and reserve space for image data
    dim format as RBGLPixelFormat = RBGLPixelFormat.RGB
    private_format = UInt32(format)
    private_width = Width
    private_height = Height
    private_actual_width = private_width
    private_actual_height = private_height
    dim rowbytes as integer = private_width * 3
    private_data = new MemoryBlock(private_height * rowbytes)

    //set alignment
    declare sub glPixelStorei lib GL_LIB (pname as Integer, param as Integer)
    glPixelStorei GL_PACK_ALIGNMENT, 1 //byte aligned

    //copy pixels from context
    declare sub glReadPixels lib GL_LIB (x as integer, y as integer, width as integer, height as integer, format as
RBGLPixelFormat, type as integer, pixels as ptr)
    glReadPixels X, Y, private_width, private_height, format, GL_UNSIGNED_BYTE, private_data

    //invert data
    for i as integer = 0 to private_height\2 - 1
        dim index1 as integer = i*rowbytes
        dim index2 as integer = (private_height - i - 1)*rowbytes
        dim s as String = private_data.StringValue(index1,rowbytes)
        private_data.StringValue(index1,rowbytes) = private_data.StringValue(index2,rowbytes)
        private_data.StringValue(index2,rowbytes) = s
    next i
End Sub
```

next

```
//upscale image to power-of-two if necessary
if private_round_to_power_of_two(private_width) > private_width or
private_round_to_power_of_two(private_height) > private_height then
```

```
    //use getpicture to extract image then use standard constructor to do scaling or padding
    Constructor GetPicture, format, Options
```

else

```
    //set texture attributes
    private_mipmapped = Options.GenerateMipMaps
    private_wrapx = Options.WrapX
    private_wrapy = Options.WrapY
    private_upscaling = RBGLScalingMode.Nearest
    private_downscaling = RBGLScalingMode.Linear
```

end

End Sub

### **RBGLPicture.DefaultOptions:**

Shared Function DefaultOptions() As RBGLPictureOptions

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
dim options as RBGLPictureOptions
options.ScaleToPowerOfTwo = False
options.WrapX = RBGLWrapMode.Clamp
options.WrapY = RBGLWrapMode.Clamp
options.Upscaling = RBGLScalingMode.Nearest
options.Downscaling = RBGLScalingMode.Linear
return options
```

End Function

### **RBGLPicture.Destructor:**

Sub Destructor()

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//init deleted textures dictionary
if private_deleted_textures_by_context = nil then
    private_deleted_textures_by_context = new Dictionary
end
```

```
//instruct all contexts to erase this texture at the next opportunity
dim handle as uint32
if private_ids_by_context <> nil then
    for each handle in private_ids_by_context.Keys
        dim id as uint32 = private_ids_by_context.Value(handle)
```

```
if private_current_context <> nil and private_current_context.Value <> nil and
RBGLContext(private_current_context.value).Handle = handle then
```

```
    //context is current – delete immediately
    declare sub glDeleteTextures lib GL_LIB (n as uint32, byref textures as uint32)
    glDeleteTextures 1, id
```

```
else
```

```
    //context is not current, add to dictionary
    if private_deleted_textures_by_context.HasKey(handle) then
```

```
        //add to existing array
        dim ids as MemoryBlock = private_deleted_textures_by_context.Value(handle)
        ids.Size = ids.Size + 4
        ids.UInt32Value(ids.Size - 4) = id
```

```
    else
```

```
        //create new array containing only me
        dim ids as new MemoryBlock(4)
        ids.UInt32Value(0) = id
        private_deleted_textures_by_context.Value(handle) = ids
```

```
    end
```

```
end
```

```
next
```

```
end
```

```
End Sub
```

### **RBGLPicture.Draw:**

```
Sub Draw(X As Single, Y As Single, Orientation As Double = 0.0)
```

```
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
    //draw image
```

```
    Draw x, y, private_width, private_height, 0, 0, private_width, private_height, Orientation
```

```
End Sub
```

### **RBGLPicture.Draw:**

```
Sub Draw(X As Single, Y As Single, Width As Single, Height As Single, Orientation As Double = 0.0)
```

```
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
    //draw image
```

```
    Draw x, y, width, Height, 0, 0, Width, Height, Orientation
```

```
End Sub
```

RBGLPicture.Draw:

```
Sub Draw(X As Single, Y As Single, Width As Single, Height As Single, SourceX As Single, SourceY as Single, SourceWidth as Single, SourceHeight As Single, Orientation As Double = 0.0)
```

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
const TOLERANCE = 0.0001
```

```
//map source coordinates to correct range
dim x1 as Double = SourceX/private_width
dim y1 as Double = SourceY/private_height
dim x2 as Double = x1 + SourceWidth/private_width
dim y2 as Double = y1 + SourceHeight/private_height
```

```
//handle padded texture coordinates
if private_padded then
    dim sx as Double = private_width/private_actual_width
    x1 = x1 * sx
    x2 = x2 * sx
    dim sy as Double = private_height/private_actual_height
    y1 = y1 * sy
    y2 = y2 * sy
end
```

```
//get current context
if private_current_context = nil or private_current_context.value = nil then return
dim context as RBGLContext = RBGLContext(private_current_context.value)
```

```
//set me as current texture
//this is faster than calling bind
context.Texture = me
```

```
//handle rotation
if abs(Orientation) > TOLERANCE then
    declare sub glPushMatrix lib GL_LIB ()
    glPushMatrix
    declare sub glTranslatef lib GL_LIB (x as Single, y as Single, z as Single)
    glTranslatef x + width\2, y + height\2, 0.0
    declare sub glRotatef lib GL_LIB (angle as Single, x as Single, y as Single, z as Single)
    glRotatef Orientation * RADIANS_TO_DEGREES, 0.0, 0.0, 1.0
    glTranslatef - (x + width\2), - (y + height\2), 0.0
end
```

```
//begin quad
declare sub glBegin lib GL_LIB (mode as UInt32)
glBegin GL_QUADS
```

```
dim xPlusWidth as Single = x + Width
dim yPlusHeight as Single = y + Height
```

```
//draw vertices
declare sub glTexCoord2d lib GL_LIB (s as Double, t as Double)
```

```

declare sub glVertex2d lib GL_LIB (x as Double, y as Double)
glTexCoord2d x1, y1
glVertex2d x, y
glTexCoord2d x2, y1
glVertex2d xPlusWidth, y
glTexCoord2d x2, y2
glVertex2d xPlusWidth, yPlusHeight
glTexCoord2d x1, y2
glVertex2d x, yPlusHeight

//end quad
declare sub glEnd lib GL_LIB ()
glEnd

//cleanup rotation
if abs(Orientation) > TOLERANCE then
    declare sub glPopMatrix lib GL_LIB ()
    glPopMatrix
end
End Sub

```

### **RBGLPicture.GetPicture:**

```

Function GetPicture() As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if private_picture = nil then
        //extract picture data
        private_get_picture_from_data private_picture, private_mask
    end

    //convert to correct size
    dim result as new Picture(private_width, private_height, 32)
    result.Graphics.DrawPicture private_picture, 0, 0, private_width, private_height
    if private_mask <> nil then
        result.mask.Graphics.DrawPicture private_mask, 0, 0, private_width, private_height
    end

    //return picture
    return result
End Function

```

### **RBGLPicture.private\_clone\_picture:**

```

Private Function private_clone_picture(Image as Picture) As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //get width and height
    dim width as integer = image.Width
    dim Height as integer = image.Height

```

```
//copy images without graphics or with a bit depth less than 32
if image.depth < 24 or image.graphics = nil or image.RGBSurface = nil then
```

```
    dim temp as Picture, mg as Graphics
    dim p as Picture = new Picture(width,height,32)
    if private_has_mask(image) then
```

```
        //preserve mask image
        dim im as Picture = image.mask
        temp = new Picture(width,height,32)
        temp.graphics.drawpicture im,0,0
```

```
        //remove image mask
        mg = im.Graphics
        mg.forecolor = &c000000
        mg.fillrect 0,0,width,height
```

```
    elseif image.Transparent = Picture.TransparentWhite then
```

```
        p.Graphics.ForeColor = &cFFFFFF
        p.Graphics.FillRect 0,0,width,height
```

```
end
```

```
//copy image
p.graphics.drawPicture image,0,0
```

```
//restore mask image
if temp <> nil then mg.drawpicture temp,0,0
p.Transparent = image.Transparent
```

```
//return sanitised image
return p
```

```
else
```

```
    //return the original image
    return image
```

```
end
```

```
End Function
```

### **RBGLPicture.private\_copy\_mac\_picture\_data:**

```
Private Sub private_copy_mac_picture_data(image as Picture, format as RBGLPixelFormat)
```

```
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false
```

```
    #if TargetMacOS
```

```
        //width and height
        dim width as uint32 = image.Width
```



```

dim height as uint32 = image.Height

//initialise variables
dim index as uint32
dim data as MemoryBlock
dim widthBound as uint32 = width - 1
dim heightBound as uint32 = height - 1

//get image data
dim rowbytes, pixelbytes as UInt32
dim d as MemoryBlock = private_get_mac_picture_data(image, rowbytes, pixelbytes)

//recode image data to match selected format
select case format

case RBGLPixelFormat.RGB

    data = newMemoryBlock(3*width*height)
    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            index = (i*width + j)*3
            data.ColorValue(index,24) = d.ColorValue(i*rowbytes + j*pixelbytes + 1, 24)
        next
    next

case RBGLPixelFormat.RGBA

    if private_has_mask(image) then

        //get mask data
        dim mrowbytes, mpixelbytes as UInt32
        dim md as MemoryBlock = private_get_mac_picture_data(image.mask, mrowbytes, mpixelbytes)

        data = newMemoryBlock(4*width*height)
        for i as uint32 = heightBound downto 0
            //copy image data
            data.StringValue(i*width*4,width*4) = d.StringValue(i*rowbytes + 1,rowbytes - 1) //data is ARGB so
            offset it by 1 for RGBA alignment
            //copy mask data
            for j as uint32 = 0 to widthBound
                index = (i*width + j)*4
                data.UInt8Value(index + 3) = 255 - md.UInt8Value(i*mrowbytes + j*mpixelbytes + 1)
            next
        next

    elseif image.Transparent = Picture.TransparentWhite then

        //use the colour white as transparent
        data = newMemoryBlock(4*width*height)
        for i as uint32 = heightBound downto 0
            //copy image data
            data.StringValue(i*width*4,width*4) = d.StringValue(i*rowbytes + 1,rowbytes - 1) //data is ARGB so
            offset it by 1 for RGBA alignment
            //copy mask data

```

```

        for j as uint32 = 0 to widthBound
            index = (i*width + j)*4
            if data.ColorValue(index, 24) = &cFFFFFF then
                data.UInt8Value(index + 3) = 0
            else
                data.UInt8Value(index + 3) = 255
            end
        next
    next
else

    //mask is opaque
    data = newMemoryBlock(4*width*height)
    for i as uint32 = heightBound downto 0
        //copy image data
        data.StringValue(i*width*4,width*4) = d.StringValue(i*rowbytes + 1,rowbytes - 1) //data is ARGB so
        offset it by 1 for RGBA alignment
        //copy mask data
        for j as uint32 = 0 to widthBound
            index = (i*width + j)*4
            data.UInt8Value(index + 3) = 255
        next
    next

end

case RBGLPixelFormat.Luminance

    data = newMemoryBlock(width*height)
    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            data.UInt8Value(i*width + j) = d.UInt8Value(i*rowbytes + j*pixelbytes + 1)
        next
    next

case RBGLPixelFormat.Alpha

    //assumes alpha is stored in main picture, not mask.
    //to extract mask alpha, pass mask as image parameter
    data = newMemoryBlock(width*height)
    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            data.UInt8Value(i*width + j) = 255 - d.UInt8Value(i*rowbytes + j*pixelbytes + 1)
        next
    next

case RBGLPixelFormat.LuminanceAlpha

    //get mask data
    dim mrowbytes, mpixelbytes as UInt32
    dim md as MemoryBlock = private_get_mac_picture_data(image.mask, mrowbytes, mpixelbytes)

    data = newMemoryBlock(2*width*height)

```

```

    for i as uint32 = heightBound DownTo 0
      for j as uint32 = 0 to widthBound
        index = (i*width + j)*2
        data.UInt8Value(index) = d.UInt8Value(i*rowbytes + j*pixelbytes + 1)
        data.UInt8Value(index + 3) = 255 - md.UInt8Value(i*mrowbytes + j*mpixelbytes + 1)
      next
    next

  else
    raise New RBGLException("Unsupported image format")
  end

  //set image attributes
  private_data = data
  private_format = UInt32(Format)
  private_actual_width = width
  private_actual_height = height

#endif
End Sub

```

### **RBGLPicture.private\_get\_mac\_picture\_data:**

Private Function private\_get\_mac\_picture\_data(image as Picture, byref rowbytes as UInt32, byref pixelbytes as UInt32) As MemoryBlock

```

#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

```

```

#if TargetMacOS

```

```

  //get the CGrafPtr for the image
  dim port as Ptr = Ptr(image.graphics.Handle(Graphics.HandleTypeCGrafPtr))

```

```

  //begin a CGContext
  dim ctx as Ptr
  declare function QDBeginCGContext lib CARBON_LIB (inPort as Ptr, byref outContext as Ptr) as OSType
  call QDBeginCGContext(port, ctx)

```

```

  //get row bytes
  declare function CGBitmapContextGetBytesPerRow lib CARBON_LIB (c as Ptr) as Integer
  rowbytes = CGBitmapContextGetBytesPerRow(ctx)

```

```

  //get bits per pixel
  declare function CGBitmapContextGetBitsPerPixel lib CARBON_LIB (c as Ptr) as Integer
  pixelbytes = CGBitmapContextGetBitsPerPixel(ctx) \ 8

```

```

  //get image data
  declare function CGBitmapContextGetData lib CARBON_LIB (c as Ptr) as Ptr
  dim data as MemoryBlock = CGBitmapContextGetData(ctx)

```

```

  //end CGContext
  declare function QDEndCGContext lib CARBON_LIB (inPort as Ptr, byref outContext as Ptr) as OSType
  call QDEndCGContext(port, ctx)

```

```
//return data
return data
```

```
#endif
```

```
End Function
```

### **RBGLPicture.private\_get\_picture\_from\_data:**

```
Private Sub private_get_picture_from_data(byref image as Picture, byref mask as Picture)
```

```
#pragma BackgroundTasks false
```

```
#pragma BoundsChecking DebugBuild
```

```
#pragma NilObjectChecking DebugBuild
```

```
#pragma StackOverflowChecking false
```

```
dim width as Integer = private_actual_width
```

```
dim height as Integer = private_actual_height
```

```
dim widthBound as uint32 = width - 1
```

```
dim heightBound as uint32 = height - 1
```

```
dim format as RBGLPixelFormat = RBGLPixelFormat(private_format)
```

```
image = new Picture(width, height, 32)
```

```
mask = nil
```

```
dim s as RGBSurface = image.RGBSurface
```

```
dim data as MemoryBlock = private_data
```

```
dim index as integer
```

```
dim a as integer
```

```
dim c as Color
```

```
//extract picture data
```

```
select case format
```

```
case RBGLPixelFormat.RGB
```

```
for i as uint32 = heightBound downto 0
```

```
for j as uint32 = 0 to widthBound
```

```
index = (i*width + j)*3
```

```
s.pixel(j,i) = rgb(data.UInt8Value(index), data.UInt8Value(index + 1), data.UInt8Value(index + 2))
```

```
next
```

```
next
```

```
case RBGLPixelFormat.RGBA
```

```
mask = new Picture(width, height, 32)
```

```
dim ms as RGBSurface = mask.RGBSurface
```

```
for i as uint32 = heightBound downto 0
```

```
for j as uint32 = 0 to widthBound
```

```
index = (i*width + j)*4
```

```
s.pixel(j,i) = rgb(data.UInt8Value(index), data.UInt8Value(index + 1), data.UInt8Value(index + 2))
```

```
a = 255 - data.UInt8Value(index + 3)
```

```
ms.pixel(j,i) = rgb(a,a,a)
```

```
next
```

```
next
```

```
case RBGLPixelFormat.Luminance
```

```

    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            a = data.UInt8Value(i*width + j)
            s.pixel(j,i) = rgb(a,a,a)
        next
    next

case RBGLPixelFormat.Alpha

    for i as uint32 = heightBound downto 0
        for j as uint32 = 0 to widthBound
            a = 255 - data.UInt8Value(i*width + j)
            s.pixel(j,i) = rgb(a,a,a)
        next
    next

case RBGLPixelFormat.LuminanceAlpha

    mask = new Picture(width, height, 32)
    dim ms as RGBSurface = mask.RGBSurface
    for i as uint32 = heightBound DownTo 0
        for j as uint32 = 0 to widthBound
            index = (i*width + j)*2
            a = data.UInt8Value(index)
            s.pixel(j,i) = rgb(a,a,a)
            a = 255 - data.UInt8Value(index + 1)
            ms.pixel(j,i) = rgb(a,a,a)
        next
    next

else
    raise New RBGLException("Unsupported image format")
end
End Sub

```

### **RBGLPicture.private\_sanitise\_picture:**

```

Private Function private_sanitise_picture(Image as Picture, ScaleToPowerOfTwo as Boolean) As Picture
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //original image size
    dim sourceWidth as uint32 = Image.Width
    dim sourceHeight as uint32 = Image.Height

    //get texture size
    dim textureWidth as uint32 = private_round_to_power_of_two(sourceWidth)
    dim textureHeight as uint32 = private_round_to_power_of_two(sourceHeight)

    //get destination size
    dim destWidth, destHeight as uint32
    if ScaleToPowerOfTwo then
        destWidth = textureWidth
    end if
end Function

```

```

    destHeight = textureHeight
else
    destWidth = sourceWidth
    destHeight = sourceHeight
end

//copy images without graphics or with a bit depth less than 32
//and/or resize image dimensions to powers of two if necessary
if image.depth < 24 or image.graphics = nil or image.RGBSurface = nil or textureWidth > sourceWidth or
textureHeight > sourceHeight then

    dim temp as Picture, mg as Graphics
    dim p as Picture = new Picture(textureWidth,textureHeight,32)
    if private_has_mask(image) then

        //clear mask
        p.mask.graphics.ForeColor = &cFFFFFF
        p.mask.graphics.FillRect 0,0,textureWidth,textureHeight

        //copy and resize mask image
        dim im as Picture = image.mask
        p.mask.graphics.drawPicture im,0,0,destWidth,destHeight,0,0,sourceWidth,sourceHeight

        //preserve mask image
        temp = new Picture(sourceWidth,sourceHeight,32)
        temp.graphics.drawpicture im,0,0

        //remove image mask
        mg = im.Graphics
        mg.forecolor = &c000000
        mg.fillrect 0,0,sourceWidth,sourceHeight

        //copy and resize image
        p.graphics.drawPicture image,0,0,destWidth,destHeight,0,0,sourceWidth,sourceHeight

        //restore mask image
        if temp <> nil then image.mask.graphics.drawpicture temp,0,0

    elseif image.Transparent = Picture.TransparentWhite then

        //create an 8-bit mask from the white pixels
        dim s as RGBSurface = image.RGBSurface
        temp = new picture(sourceWidth,sourceHeight, 32)
        if image.depth < 24 or s = nil then
            //image has no rgburface, copy into temp instead
            temp.Graphics.DrawPicture image, 0,0,destWidth,destHeight,0,0,sourceWidth,sourceHeight
            s = temp.RGBSurface
        end

        dim d as RGBSurface = p.mask.RGBSurface
        if ScaleToPowerOfTwo then
            //mask needs to be scaled, so draw into temp first
            d = temp.RGBSurface
        else

```

```

        //clear mask
        p.mask.graphics.ForeColor = &cFFFFFF
        p.mask.graphics.FillRect 0,0,textureWidth,textureHeight
    end

    //convert image to mask
    for i as uint32 = sourceHeight - 1 downto 0
        for j as uint32 = sourceWidth - 1 downto 0
            if s.pixel(j,i) <> &cFFFFFF then
                d.Pixel(j,i) = &c000000
            end
        next
    next

    if ScaleToPowerOfTwo then
        //copy and resize mask
        p.mask.graphics.drawPicture temp,0,0,textureWidth,textureHeight,0,0,sourceWidth,sourceHeight
    end

    //copy and resize image
    p.graphics.drawPicture image,0,0,destWidth,destHeight,0,0,sourceWidth,sourceHeight

else

    //copy and resize image
    p.graphics.drawPicture image,0,0,destWidth,destHeight,0,0,sourceWidth,sourceHeight

end

//return sanitised image
return p

else

    //return the original image
    return image

end
End Function

```

### **RBGLPicture.Tile:**

```

Sub Tile(X As Single, Y As Single, Width As Single, Height As Single, OffsetX As Single = 0, OffsetY As Single = 0)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    if not private_padded then

        //map source coordinates to correct range
        dim x1 as Single = OffsetX/private_width
        dim y1 as Single = OffsetY/private_height
        dim x2 as Single = x1 + width/private_width
        dim y2 as Single = y1 + height/private_height
    end if
end Sub

```

```

//get current context
if private_current_context = nil or private_current_context.value = nil then return
dim context as RBGLContext = RBGLContext(private_current_context.value)

//get current texture
dim texture as RBGLPicture = context.Texture

//set me as current texture
context.Texture = me

//set wrap mode (ignore texture settings)
declare sub glTexParameterI lib GL_LIB (target as uint32, pname as uint32, param as RBGLWrapMode)
glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, RBGLWrapMode.Repeat
glTexParameterI GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, RBGLWrapMode.Repeat

//begin quad
declare sub glBegin lib GL_LIB (mode as UInt32)
glBegin GL_QUADS

dim xPlusWidth as Single = x + Width
dim yPlusHeight as Single = y + Height

//draw vertices
declare sub glTexCoord2f lib GL_LIB (s as Single, t as Single)
declare sub glVertex2f lib GL_LIB (x as Single, y as Single)
glTexCoord2f x1, y1
glVertex2f x, y
glTexCoord2f x2, y1
glVertex2f xPlusWidth, y
glTexCoord2f x2, y2
glVertex2f xPlusWidth, yPlusHeight
glTexCoord2f x1, y2
glVertex2f x, yPlusHeight

//end quad
declare sub glEnd lib GL_LIB ()
glEnd

//restore previous texture
context.Texture = texture
else
if offsetX > 0 then
    offsetX = offsetX - floor(offsetX/private_width)*private_width
else
    offsetX = offsetX - ceil(offsetX/private_width)*private_width
end

if offsetY > 0 then
    offsetY = offsetY - floor(offsetY/private_height)*private_height
else
    offsetY = offsetY - ceil(offsetY/private_height)*private_height

```



end

//horizontal metrics

```
dim bl as single = private_width - offsetX
dim w as single = width - bl
dim tilex as single = floor(w/private_width)
dim br as single = w - tilex*private_width
dim blo as single = X + bl
dim bro as single = X + width - br
```

//vertical metrics

```
dim bt as single = private_height - offsetY
dim h as single = height - bt
dim tiley as single = floor(h/private_height)
dim bb as single = h - tiley*private_height
dim bto as single = Y + bt
dim bbo as single = Y + height - bb
```

h = h - bb

//corners

```
Draw X, Y, bl, bt, offsetX, offsetY, bl, bt
Draw bro, Y, br, bt, 0, offsetY, br, bt
Draw bro, bbo, br, bb, 0, 0, br, bb
Draw X, bbo, bl, bb, offsetX, 0, bl, bb
```

//horizontal edges

```
for i as integer = 0 to tilex
    dim offset as single = blo + i*private_width
    Draw offset, Y, private_width, bt, 0, offsetY, private_width, bt
    Draw offset, bbo, private_width, bb, 0, 0, private_width, bb
next
```

```
for i as integer = 0 to tiley
    dim offset as single = bto + i*private_height
```

//vertical edges

```
Draw X, offset, bl, private_height, offsetX, 0, bl, private_height
Draw bro, offset, br, private_height, 0, 0, br, private_height
```

//center tiles

```
for j as integer = 0 to tilex
    Draw blo + j*private_width, offset
next
```

next

end

End Sub

## **RBGLPicture.ActualHeight:**

ActualHeight As UInt32

Get

```

        return private_actual_width
    End Get
End Property

```

## **RBGLPicture.ActualWidth:**

ActualWidth As UInt32

```

    Get
        return private_actual_height
    End Get
End Property

```

## **RBGLPicture.Downscaling:**

Downscaling As RBGLScalingMode

```

    Get
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        return private_downscaling
    End Get
    Set
        #pragma BackgroundTasks false
        #pragma BoundsChecking DebugBuild
        #pragma NilObjectChecking DebugBuild
        #pragma StackOverflowChecking false

        //update value
        private_downscaling = value

        //if currently bound...
        if private_current_context <> nil and private_current_context.value <> nil and
            RBGLContext(private_current_context.value).Texture = me then

            //set filtering mode
            declare sub glTexParameteri lib GL_LIB (target as uint32, pname as uint32, param as uint32)
            if private_mipmapped then
                if private_downscaling = RBGLScalingMode.Linear then
                    glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR
                else
                    glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR
                end
            else
                glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, UInt32(private_downscaling)
            end
            glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, UInt32(private_upscaling)

        end
    End Set
End Property

```

RBGLPicture.Height:

Height As UInt32

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_height
```

End Get

End Property

**RBGLPicture.Opaque:**

Opaque As Boolean

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//return true if image is opaque (doesn't have a mask)
return private_format = Integer(RBGLPixelFormat.RGB)
```

End Get

End Property

Private private\_actual\_height As UInt32

Private private\_actual\_width As UInt32

Private private\_data As MemoryBlock

Private private\_downscaling As RBGLScalingMode

Private private\_format As uint32

Private private\_height As uint32

Private private\_ids\_by\_context As Dictionary

Private private\_mask As Picture

Private private\_mipmapped As Boolean

Private private\_padded As Boolean

Private private\_picture As Picture

Private private\_upscaling As RBGLScalingMode

Private private\_width As uint32

Private private\_wrapx As RBGLWrapMode

Private private\_wrapy As RBGLWrapMode

## **RBGLPicture.Upscaling:**

Upscaling As RBGLScalingMode

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

return private\_upscaling

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
//update value
private_upscaling = value
```

```
//if currently bound...
```

```
if private_current_context <> nil and private_current_context.value <> nil and
RBGLContext(private_current_context.value).Texture = me then
```

```
//set filtering mode
```

```
declare sub glTexParameteri lib GL_LIB (target as uint32, pname as uint32, param as RBGLScalingMode)
gltexParameteri GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, private_upscaling
```

end

End Set

End Property

## **RBGLPicture.Width:**

Width As UInt32

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

return private\_width

End Get

End Property

## **RBGLPicture.WrapX:**

WrapX As RBGLWrapMode

Get

```
#pragma BackgroundTasks false
```

```

#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

return private_wrapx
End Get
Set
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//update value
private_wrapx = value

//if currently bound...
if private_current_context <> nil and private_current_context.value <> nil and
RBGLContext(private_current_context.value).Texture = me then

    //set x wrap mode
    declare sub glTexParameteri lib GL_LIB (target as uint32, pname as uint32, param as RBGLWrapMode)
    glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, private_wrapx

end
End Set
End Property

```

## **RBGLPicture.WrapY:**

WrapY As RBGLWrapMode

```

Get
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

return private_wrapy
End Get
Set
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

//update value
private_wrapy = value

//if currently bound...
if private_current_context <> nil and private_current_context.value <> nil and
RBGLContext(private_current_context.value).Texture = me then

    //set y wrap mode
    declare sub glTexParameteri lib GL_LIB (target as uint32, pname as uint32, param as RBGLWrapMode)
    glTexParameteri GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, private_wrapx

```

```

        end
    End Set
End Property
End Class
Class RBGLStringTexture

```

### **RBGLStringTexture.Blit:**

```

Sub Blit(X As Integer, Y As Integer)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //blit image
    dim offset as Single = LineHeight/2
    Texture.Blit x - offset, y - offset - private_graphics.TextAscent
End Sub

```

### **RBGLStringTexture.Constructor:**

```

Sub Constructor(Text as String)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //prevent re-initialisation
    if private_picture <> nil then
        return
    end

    //set string
    private_text = Text

    //allocate minimal picture storage
    private_picture = NewPicture(1,1,32)

    //set graphics attribute - used for text layout
    private_graphics = private_picture.Graphics
End Sub

```

### **RBGLStringTexture.Draw:**

```

Sub Draw(X As Single, Y As Single)
    #pragma BackgroundTasks false
    #pragma BoundsChecking DebugBuild
    #pragma NilObjectChecking DebugBuild
    #pragma StackOverflowChecking false

    //draw texture
    dim offset as Single = LineHeight/2
    Texture.Draw x - offset, y - offset - private_graphics.TextAscent
End Sub

```

## **RBGLStringTexture.Ascent:**

Ascent As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.TextAscent
```

End Get

End Property

## **RBGLStringTexture.Bold:**

Bold As Boolean

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.Bold
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if value <> private_graphics.Bold then
    private_graphics.Bold = value
    private_texture = nil
end
```

End Set

End Property

## **RBGLStringTexture.Condense:**

Condense As Boolean

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_condense
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
```

```

#pragma StackOverflowChecking false

if value <> private_condense then
  private_condense = value
  private_texture = nil
end
End Set
End Property

```

### **RBGLStringTexture.Descent:**

Descent As Integer

```

Get
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild
  #pragma StackOverflowChecking false

  return private_graphics.TextHeight - private_graphics.TextAscent
End Get
End Property

```

### **RBGLStringTexture.FontName:**

FontName As String

```

Get
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild
  #pragma StackOverflowChecking false

  return private_graphics.TextFont
End Get
Set
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild
  #pragma StackOverflowChecking false

  if value <> private_graphics.TextFont then
    private_graphics.TextFont = value
    private_texture = nil
  end
End Set
End Property

```

### **RBGLStringTexture.FontSize:**

FontSize As Integer

```

Get
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild

```



```

#pragma StackOverflowChecking false

return private_graphics.TextSize
End Get
Set
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

if value <> private_graphics.TextSize then
    private_graphics.TextSize = value
    private_texture = nil
end
End Set
End Property

```

### **RBGLStringTexture.Height:**

Height As Integer

```

Get
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

dim w as integer = private_wrap_width
if w = 0 then w = 10000
return private_graphics.StringHeight(private_text,w)
End Get
End Property

```

### **RBGLStringTexture.Italic:**

Italic As Boolean

```

Get
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

return private_graphics.Italic
End Get
Set
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false

if value <> private_graphics.Italic then
    private_graphics.Italic = value
    private_texture = nil
end
End Set

```

End Property

### **RBGLStringTexture.LineHeight:**

LineHeight As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.TextHeight
```

End Get

End Property

Private private\_condense As Boolean

Private private\_graphics As Graphics

Private private\_picture As Picture

Private private\_text As String

Private private\_texture As RBGLPicture

Private private\_wrap\_width As Integer

### **RBGLStringTexture.Text:**

Text As String

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_text
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if value <> private_text then
    private_text = value
    private_texture = nil
end
```

End Set

End Property

### **RBGLStringTexture.TextDirection:**

TextDirection As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.StringDirection(private_text)
```

End Get

End Property

## **RBGLStringTexture.Texture:**

Texture As RBGLPicture

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if private_texture = nil then
```

```
    //need to generate text image
    dim w as integer = ceil(Width) + LineHeight
    dim tw as integer = private_round_to_power_of_two(w)
    dim h as integer = Height + LineHeight
    dim th as integer = private_round_to_power_of_two(h)
```

```
    //create text image
    dim p as Picture = NewPicture(tw,th,32)
```

```
    //copy text attributes to new image
    dim tg as Graphics = p.Graphics
    tg.TextFont = private_graphics.TextFont
    tg.TextSize = private_graphics.TextSize
    tg.Bold = private_graphics.Bold
    tg.Italic = private_graphics.Italic
    tg.Underline = private_graphics.Underline
```

```
    //draw text
    tg.ForeColor = &c000000
    tg.DrawString private_text, LineHeight/2, LineHeight/2 + tg.TextAscent, private_wrap_width,
    private_condense
```

```
    //create texture
    dim options as RBGLPictureOptions = RBGLPicture.DefaultOptions
    options.Upscaling = RBGLScalingMode.Nearest
    private_texture = new RBGLPicture(p, RBGLPixelFormat.Alpha, options)
```

```
end
```

```
return private_texture
```

End Get

End Property

## **RBGLStringTexture.Underline:**

Underline As Boolean

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_graphics.Underline
```

End Get

Set

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if value <> private_graphics.Underline then
    private_graphics.Underline = value
    private_texture = nil
end
```

End Set

End Property

## **RBGLStringTexture.Width:**

Width As Double

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
if wrapWidth = 0 then
    return private_graphics.StringWidth(private_text)
else
    return min(private_graphics.StringWidth(private_text), private_wrap_width)
end
```

End Get

End Property

## **RBGLStringTexture.WrapWidth:**

WrapWidth As Integer

Get

```
#pragma BackgroundTasks false
#pragma BoundsChecking DebugBuild
#pragma NilObjectChecking DebugBuild
#pragma StackOverflowChecking false
```

```
return private_wrap_width
```

End Get

```

Set
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild
  #pragma StackOverflowChecking false

  if value <> private_wrap_width then
    private_wrap_width = value
    private_texture = nil
  end
End Set
End Property
End Class
Class RBGLException
Inherits RuntimeException

```

### **RBGLException.Constructor:**

```

Sub Constructor(message as String)
  #pragma BackgroundTasks false
  #pragma BoundsChecking DebugBuild
  #pragma NilObjectChecking DebugBuild
  #pragma StackOverflowChecking false

  me.Message = Message
End Sub
End Class
End Module

```